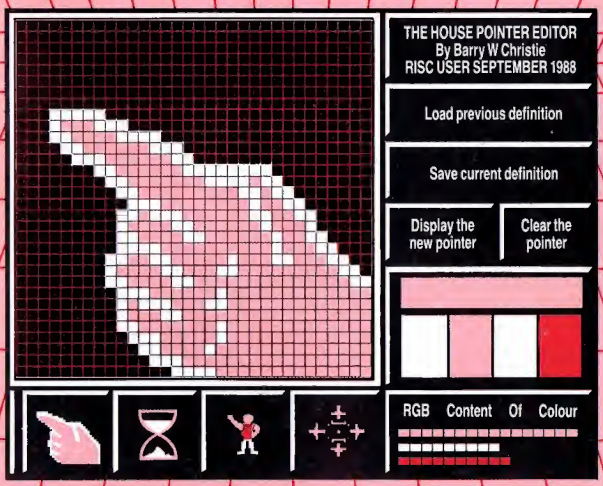
Volume 1
Issue 10

October
1988

Price £1.20

RISC USER

Risc User Pointer Definer



THE MAGAZINE AND SUPPORT GROUP
EXCLUSIVELY FOR USERS OF THE ARCHIMEDES

Volume 1 Issue 10
October 1988

RISC USER

CONTENTS

RISC User is published by BEEBUG Ltd.
Co-Editors: Mike Williams, Dr Lee Calcraft
Assistant Editor: Kristina Lucas
Technical Editor: David Spencer
Production Assistant: Yolanda Turuelo
Technical Assistant: Lance Allison
Subscriptions: Mandy Mileham

BEEBUG, Dolphin Place, Holywell Hill, St.Albans, Herts AL1 1EX. Tel. St.Albans (0727) 40303

Printed by Arlon House

The Archimedes Magazine
and Support Group.

EDITORIAL

This issue of RISC User completes volume one, and to commemorate our first birthday, we have prepared a **special disc** packed with goodies, which will be available to all valid members as at 1 November 1988 (see page 5 for further details).

Amongst other things the disc contains a piece of software called ArcScan. This is a mouse-driven bibliographic program for locating items from RISC User. It comes with data for the first volume of RISC User, and for the first six volumes of BEEBUG magazine. Each RISC User monthly disc will contain data updates for that issue; and for those not subscribing to the monthly discs, we shall also be producing annual updates.

ArcScan permits searching on up to two strings, with selectable logic and both asterisk and hash wildcards. Each record in the data contains the title of the item plus a set of keywords to make searching easy. This makes it a simple matter to locate any hint, review or program. For example, searching on the word MOUSE will find all articles, programs and hints relating to the mouse. The search routines are written in ARM code, and therefore operate at very high speed. The 120K BEEBUG database is searched in less than one second. As a further bibliographic aid, next month's magazine will contain a printed index for volume one.

We hope that you have found RISC User volume one to be indispensable reading, and that volume two will prove to be even more so.

*This month's Telesoftware password is **blackbird.***
(see BEEBUG pages on Micronet)

PODULES AND BACKPLANES

IFEL has just launched a four slot backplane for 300 series machines, as an alternative to Acorn's two slot option. The backplane, which comes complete with all the necessary fittings, and a cooling fan, costs £59.95 inc. VAT, and is available from IFEL, 36 Upland Drive, Derriford, Plymouth PL6 6BD, tel. (07555) 7286.

Once you have got your backplane, SGB computer services can sell you a buffer podule for £49.50 inc. VAT, allowing other podules to be mounted outside the computer. This simplifies the development of new podules, and also allows extra large podules to be fitted. SGB Computer Services are at 140 Disraeli Road, London SW15 2DX, tel. 01-874 5675.

A NEW DIMENSION FOR SPREADSHEETS

Matrix-3 from Cambridge Microsystems is a new concept in spreadsheets. Unlike most packages, Matrix-3 is three dimensional, with pages as well as rows and columns. A large number of mathematical functions are provided, and these can be combined to form not only simple expressions, but also complete programs which can then be stored in a single cell. The display can show two windows, each with a different area of the spreadsheet. There are also a number of graph plotting features, and a complete help system. The package costs £109.25 inc. VAT from Cambridge Microsystems, 19 Panton Street, Cambridge CB2 1HL, tel. (0223) 66553.

ALTERNATIVE HARD DISC

Watford Electronics is producing a hard disc upgrade for the Archimedes, as a direct rival to Acorn's own version. The upgrade will consist of an internally fitted 3.5" Winchester drive and a controller podule, which must be fitted to a podule backplane, much the same as the Acorn upgrade. The Watford hard disc will be available in both 20Mbyte and 40Mbyte versions (Acorn offers 20Mbyte only). The upgrades should be available sometime in October. While the price has yet to be fixed, Nazir Jessa says he expects the 40 Mbyte unit to be £100 less than Acorn's 20 Mbyte device (which is £575 inc. VAT). For more details, contact Watford Electronics, Jessa House, 250 Lower High St., Watford WD1 2AN, tel. (0923) 37774.

CIRCUIT BOARD DESIGN

ARC-PCB from Silicon Vision is a complete Printed Circuit Board (PCB) designing program. The system allows PCBs of up to eight layers to be drawn to a resolution of 1 Mil (1 thousandth of an inch). Each board can contain as many as 120,000 components, including surface mounted devices, with a facility for component libraries to be built up. The design can then be printed on an Epson printer, or a plotter, ready to produce the actual PCB. ARC-PCB costs £99.95 inc. VAT, or £195 inc. VAT for a version with automatic track routing. A demo disc is available for £5. Silicon Vision also offer complete PCB design systems including the software, an Archimedes, and a plotter. More information from Silicon Vision Ltd, Signal House, Lyon Road, Harrow, Middlesex HA1 2AG, tel. 01-422 2274.

DFS READER

New from BEEBUG is a package which enables DFS format files to be transferred to and from an Archimedes (for example using an external 5.25" disc drive). The reader software is in the form of a relocatable module, and will handle not only the Acorn DFS format discs, but also those produced with the Watford DFS and DDFS. The DFS reader costs £9.90 inc. VAT to RISC User members, while BEEBUG's 5.25" interface is £27.60 inc. VAT. Both can be obtained from the address given on the back cover.

PIPEDREAM SPELLING CHECKER

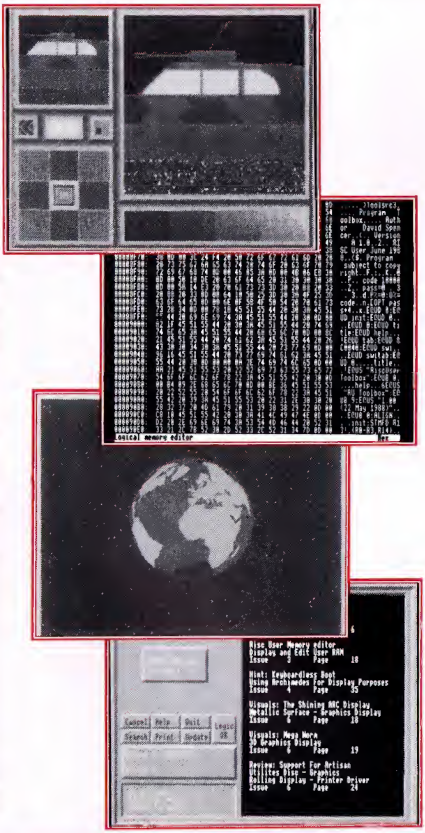
To complement its Pipedream integrated word processor and spreadsheet (see RISC User Issue 8), Colton Software is releasing a spelling checker. The package will have a built in dictionary of between 80,000 and 100,000 words, and any number of user dictionaries can be set up. There is an option to check text as it is typed in, or to check an entire document, in which case speeds of about 40,000 wpm are claimed. Pipedream Spellcheck should be available around mid-October at under £50. A voucher is also included to enable existing users to make the essential upgrade to the latest version 2.2 of Pipedream free of charge. More details from Colton Software, Broadway House, 149-151 St Neots Road, Cambridge CB3 7QJ, tel. (0954) 211472.

RU

RISC USER

The special disc to celebrate our 1st birthday will be available with the November issue.

SPECIAL DISC CONTAINS ALL THIS FOR JUST 4.95



1. ARCSCAN
   A fast on-screen bibliography with powerful search facilities for all the RISC User and BEEBUG magazines. Normal price £12.

2. PIXEL EDITOR
   This powerful drawing tool is a full screen full-feature pixel editor for creating and editing screens and sprites.

3. TOOLBOX
   This incredibly useful utility features a memory editor, memory search and replace, disc editor and disassembler. TOOLBOX contains many of the features found in packages costing over £35.

4. WORLD IN MOTION
   A stunning animation with an oddly reminiscent feel to it.

5. DISC MENU MODULE
   Use the mouse to control your disc files with this extremely useful relocatable module.

6. PRINTER BUFFER
   This printer buffer frees your computer during long printouts and is configurable from a few bytes to 4 Mbytes. Similar to packages currently selling at £19.

Altogether the items on the disc would be worth over £50 if bought separately. See your subscription reminder or next month's magazine for full details.

A REAL-TIME IMAGE SPINNER

Use this piece of code by Stephen Streater to spin images around the screen in real-time.

Because the ARM is so stunningly fast it is possible to rotate screen images in real-time on the Archimedes. The accompanying program does just that for any image in mode 12 or 13. When it is run it first loads a screen image (called SCREEN) from disc, and transfers it to a work area above the program. It then continually rotates and resizes the image in such a way that the picture appears from a point at the centre of the screen, rotating and growing in size until it occupies about half of the screen.

Precisely how the image is manipulated depends on the parameters of the procedure PROCrot in line 140. There are five, as follows: the x and y co-ordinates of the centre of the displayed picture (these remain fixed in the example), the height and width of the displayed picture, and the angle of rotation. The first four are given in graphics units, while the fifth is given in radians, and must be greater than or equal to zero, and less than 2*PI. To make the image spin, as we have done in the example, just requires calling PROCrot within a loop which progressively alters the angle of display.

The program is relatively long because it contains eight separate rotation routines; the one used on any given occasion will depend (automatically) upon the required angle of rotation. Remember too that the routine simultaneously rescales the image, as well as rotating it. In order to keep the program to a reasonable length, a number of short cuts have been made - for example there is no checking for images which overlap the screen edge, so you must take care not to let the image grow too large. Next month we will add a routine to cope with this. Part 2 will also contain other enhancements which improve the quality of the image generated and the variety of effects which can be produced.

```
 10 REM             >Spinner
 20 REM Program     Image Spinner
 30 REM Version     A 0.6
 40 REM Author      S.B.Streater
 50 REM RISC User October 1988
 60 REM Program     Subject to Copyright
 70 :
 80 DIM s% 4000+80*1024:CLS
 90 PROCasm(0,s%):PROCasm(2,s%)
100 *ScreenLoad Screen
110 OFF:CALL copy:CLS
120 REPEAT:Z=628.2/1.01^200
130 REPEAT:Z=Z*1.01
140 PROCrot(640,512,Z,Z,2*PI-Z/100)
150 UNTILZ>=628
160 UNTIL FALSE
170 :
180 DEF PROCasm(Z,P%)
190 [OPT Z:.input   EQUD 148:EQUD TRUE
200 .output  EQUD 0
210 .copy:ADR   R0, input
220    ADR   R1, output
230    SWI   "OS_ReadVduVariables"
240    LDR   R0, output
250    LDR   R1, workspace1
260    MOV   R2, #80*1024
270 .copy_1:LDMIA R0!, {R3-R12}
280 STMIA R1!,{R3-R12}:SUBSR2,R2,#40
290    BNE   copy_1:MOV   PC, R14
300 .x_begin EQUD 0:.y_begin EQUD 0
310 .x1 EQUD 0:.y1 EQUD 0
320 .x2 EQUD 0:.y2 EQUD 0
330 .x3 EQUD 0:.x4 EQUD 0
340 .temp  EQUD 0:EQUD 0:EQUD 0:EQUD 0
350 .t_       EQUD temp
360 .link  EQUD 0:.stack EQUD 0
370 .resize_1:FN_init_resize
380    MOV R9, #0:MOV R10, #255<<16
390    LDR   R8, y2:FN_r_B(1)
400    CMP   R0, #320<<16:BLE   e_1
410    SUB   R9, R0, R6
420    ADD   R10, R1, R7
```

```
430     ADD   R10, R10, R8
440     SUB   R11, R2, #1<<16
450     ADD   R12, R3, #1<<16
460 .r_2:FN_r_A(0):ADD   R10, R10, R8
470     SUB   R11, R11, R14
480     ADD   R12, R12, #1<<16
490     CMP   R10, #256<<16:BLT   r_2
500     LDR R11,x_begin:LDR R12,y_begin
510     ADD   R11, R11, R14
520     SUB   R12, R12, #1<<16
530     MOV   R9, #0
540     RSB   R10, R8, #255<<16
550 .r_4:FN_r_B(1):SUBS  R10, R10, R8
560     ADD   R11, R11, R14
570     SUB   R12, R12, #1<<16
580     BGE   r_4:B     endif_1
590 .e_1:LDR R4,t_:STMIA R4,{R0-R3}
600     LDR   R14, x4:LDR   R8, x2
610     ADD   R9, R9, R8
620     ADD   R11, R11, R14
630     ADD   R12, R12, #1<<16
640 .r_6:FN_r_B(1):ADD   R9, R9, R8
650     ADD   R11, R11, R14
660     ADD   R12, R12, #1<<16
670     CMP   R9, #320<<16:BMI   r_6
680     LDR   R4, t_:LDMIA R4, {R9-R12}
690     SUB   R11, R11, #1<<16
700     SUB   R11, R11, R14
710     SUB   R12, R12, #1<<16
720     SUB   R9, R9, R8
730     SUB   R9, R9, R6
740     ADD   R10, R10, R7
750 .r_8:FN_r_A(0):SUBS  R9, R9, R8
760     SUB   R11, R11, R14
770     SUB   R12, R12, #1<<16
780     BGE   r_8
790 .endif_1:LDR   R13, stack
800     LDR   PC, link
810 .resize_2:FN_init_resize
820 MOV R9, #320<<16:MOV R10, #255<<16
830     SUB   R9, R9, #1<<16
840     LDR   R8, x2:FN_r_D(1)
850     CMP   R0, #0:BLE   e_1B
860     ADD   R9, R0, R6
870     ADD   R10, R1, R7
880     ADD   R9, R9, R8
890     SUB   R11, R2, #1<<16
900     SUB   R11, R11, R14
910     ADD   R12, R3, #1<<16
920 .r_2B:FN_r_C(0):ADD   R9, R9, R8
930     SUB   R11, R11, R14
940     ADD   R12, R12, #1<<16
950     CMP   R9, #320<<16:BLT   r_2B
960     LDR R11,x_begin:LDR R12,y_begin
970     ADD   R11, R11, R14
980     SUB   R12, R12, #1<<16
990     RSB   R9, R8, #320<<16
1000    SUB   R9, R9, #1<<16
1010    MOV   R10, #255<<16
1020 .r_4B:FN_r_D(1):SUBS  R9, R9, R8
1030    ADD   R11, R11, R14
1040    SUB   R12, R12, #1<<16
1050    BGE   r_4B:B     endif_1
1060 .e_1B:LDR R4,t_:STMIA R4,{R0-R3}
1070    LDR   R8, y2:LDR   R14, x4
1080    SUB   R10, R10, R8
1090    ADD   R11, R11, R14
1100    ADD   R12, R12, #1<<16
1110 .r_6B:FN_r_D(1):SUBS  R10, R10, R8
1120    ADD   R11, R11, R14
1130    ADD   R12, R12, #1<<16
1140    BGE   r_6B
1150    LDR   R4, t_:LDMIA R4, {R9-R12}
1160    SUB   R12, R12, #1<<16
1170    SUB   R11, R11, #1<<16
1180    ADD   R9, R9, R6
1190    ADD   R10, R10, R7
1200    ADD   R10, R10, R8
1210 .r_8B:FN_r_C(0):ADD   R10, R10, R8
1220    SUB   R11, R11, R14
1230    SUB   R12, R12, #1<<16
1240    CMP   R10, #256<<16:BLT   r_8B
1250    B     endif_1
1260 .resize_3:FN_init_resize
1270    MOV R9, #320<<16:MOV R10, #0
1280    SUB   R9, R9, #1<<16
1290    LDR   R8, y2:FN_r_A(1)
1300    CMP   R0, #0:BGE   e_1C
1310    ADD   R9, R0, R6
1320    SUB   R10, R1, R7
1330    SUB   R10, R10, R8
1340    SUB   R11, R2, #1<<16
1350    SUB   R11, R11, R14
1360    ADD   R12, R3, #1<<16
1370 .r_2C:FN_r_B(0):SUBS  R10, R10, R8
1380    SUB   R11, R11, R14
1390    ADD   R12, R12, #1<<16
1400    BGE   r_2C
1410    LDR R11,x_begin:LDR R12,y_begin
1420    ADD   R11, R11, R14
1430    SUB   R12, R12, #1<<16
1440    MOV R9, #320<<16:MOV R10, R8
1450    SUB   R9, R9, #1<<16
1460 .r_4C:FN_r_A(1):ADD   R10, R10, R8
1470    ADD   R11, R11, R14
1480    SUB   R12, R12, #1<<16
```

```
1490    CMP   R10, #256<<16:BLT   r_4C
1500    B     endif_1
1510 .e_1C:LDR R4,t_:STMIA R4,{R0-R3}
1520    LDR   R14, x4:LDR   R8, x2
1530    SUB   R9, R9, R8
1540    ADD   R11, R11, R14
1550    ADD   R12, R12, #1<<16
1560 .r_6C:FN_r_A(1):SUBS  R9, R9, R8
1570    ADD   R11, R11, R14
1580    ADD   R12, R12, #1<<16
1590    BGE   r_6C
1600    LDR   R4, t_:LDMIA R4, {R9-R12}
1610    SUB   R11, R11, #1<<16
1620    SUB   R12, R12, #1<<16
1630    ADD   R9, R9, R6
1640    ADD   R9, R9, R8
1650    SUB   R10, R10, R7
1660 .r_8C:FN_r_B(0):ADD   R9, R9, R8
1670    SUB   R11, R11, R14
1680    SUB   R12, R12, #1<<16
1690    CMP   R9, #320<<16:BLT   r_8C
1700    B     endif_1
1710 .resize_4:FN_init_resize
1720    MOV   R9, #0:MOV   R10, #0
1730    LDR   R8, y2:LDR   R14, x4
1740    FN_r_C(1)
1750    CMP   R0, #320<<16:BLT   e_1D
1760    SUB   R9, R0, R6
1770    SUB   R10, R1, R7
1780    SUB   R10, R10, R8
1790    SUB   R11, R2, #1<<16
1800    SUB   R11, R11, R14
1810    SUB   R12, R3, #1<<16
1820 .r_2D:FN_r_D(0):SUBS  R10, R10, R8
1830    SUB   R11, R11, R14
1840    SUB   R12, R12, #1<<16
1850    BGE   r_2D
1860    LDR R11,x_begin:LDR R12,y_begin
1870    ADD   R11, R11, R14
1880    ADD   R12, R12, #1<<16
1890    MOV   R9, #0:MOV   R10, R8
1900 .r_4D:FN_r_C(1):ADD   R10, R10, R8
1910    ADD   R11, R11, R14
1920    ADD   R12, R12, #1<<16
1930    CMP   R10, #256<<16:BLT   r_4D
1940    B     endif_1
1950 .e_1D:LDR R4,t_:STMIA R4,{R0-R3}
1960    LDR   R8, x2:LDR   R14, x3
1970    ADD   R9, R9, R8
1980    ADD   R11, R11, R14
1990    SUB   R12, R12, #1<<16
2000 .r_6D:FN_r_C(1):ADD   R9, R9, R8
2010    ADD   R11, R11, R14
2020    SUB   R12, R12, #1<<16
2030    CMP   R9, #320<<16:BLT   r_6D
2040    LDR   R4, t_:LDMIA R4, {R9-R12}
2050    ADD   R12, R12, #1<<16
2060    SUB   R11, R11, #1<<16
2070    SUB   R9, R9, R6
2080    SUB   R10, R10, R7
2090 .r_8D:FN_r_D(0):SUBS  R9, R9, R8
2100    SUB   R11, R11, R14
2110    ADD   R12, R12, #1<<16
2120    BGE   r_8D:B     endif_1
2130 .workspace1 EQUD workspace
2140 .workspace:]:P%+=80*1024
2150 ENDPROC
2160 :
2170 DEF FN_plot
2180 [OPT Z:LDR   R5, workspace1
2190    MOV   R4, R1, LSR #16
2200    ADD   R4, R4, R4, LSL #2
2210    ADD   R4, R5, R4, LSL #6
2220    LDRB  R4, [R4, R0, LSR #16]
2230    MOV   R5, R3, LSR #16
2240    ADD   R5, R5, R5, LSL #2
2250    ADD   R5, R13, R5, LSL #6
2260    STRB  R4, [R5, R2, LSR #16]
2270 ]:IF a THEN
2280 [OPT Z:ADD   R2, R2, #1<<16:]
2290 ELSE
2300 [OPT Z:SUB   R2, R2, #1<<16:]
2310 ENDIF:=0
2320 DEF FN_init_loop
2330 [OPT Z:MOV   R0, R9:MOV   R1, R10
2340    MOV   R2, R11:MOV   R3, R12
2350 ]:=0
2360 :
2370 DEF FN_r_A(a)
2380 [OPTZ:FN_init_loop:.repeat:FN_plot
2390    SUBS  R0, R0, R6:ADD  R1,R1,R7
2400    ADDLT R15, R15, #4
2410    CMP   R1, #256<<16:BLT   repeat
2420 ]:=0
2430 :
2440 DEF FN_r_B(a)
2450 [OPTZ:FN_init_loop:.repeat:FN_plot
2460    ADD   R0, R0, R6:SUBS R1,R1,R7
2470    ADDLT R15, R15, #4
2480    CMP   R0, #320<<16:BLT   repeat
2490 ]:=0
2500 :
2510 DEF FN_r_C(a)
2520 [OPTZ:FN_init_loop:.repeat:FN_plot
2530    ADD   R0, R0, R6:ADD  R1,R1,R7
```

Continued on page 12

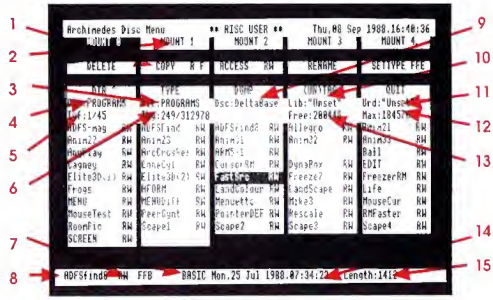
SUPERCHARGE THE RISC USER DISC MENU

David Pilling adds a host of useful features to his very popular disc front end.

If you carefully add the accompanying listing to the Disc Menu program from Issue 2 of RISC User, taking great care not to renumber any of the lines, you will end up with the new version. When you run this, it will create a relocatable module, and will automatically save it to disc under the name RMENU. To use the module, type:

```
QUIT
RMENU
```

Typing *MENU at any time will then invoke the menu. To quit, press Escape, or use the "Quit" box.



FEATURES of the Supercharged Menu

1. New Copy Options
2. Current Drive Highlighted
3. Directory Title
4. Directory Name
5. Number of Tagged and Untagged Files
6. Bytes used by Tagged Files and Bytes used by Current Directory
7. Type of File Under Pointer
8. Name of File Under Pointer
9. Disc Name
10. Library
11. User Root Directory
12. Largest Free Area on Disc
13. Total Free Space on Disc
14. Date of File under Pointer
15. Length of File under Pointer

You will gather most of the new features from the illustration. The major ones are as follows:

1. The current drive is highlighted.
2. The number of free bytes on the disc is displayed.
3. The number of bytes used in the current directory is displayed.
4. The largest free area of the disc is displayed (giving the size of the largest file that can be saved before compacting).
5. The default COPY options have been altered to "R" (Recurse) and "F" (Force), while "P" (Prompting) is only selected on single-drive systems.
6. The file length at the base of the screen is now in decimal rather than hex.
7. The "UNTAG" option has been altered to allow tagging or untagging of all files.
8. The number of tagged and untagged files is now displayed.
9. Colour numbers have been redefined in a way which permits the palette to be adjusted more effectively prior to assembly.

Next month we will briefly explain how to alter the palette, and provide an alternative colour scheme for the menu.

```
  3 REM >MENUdiff
 316  BL drives
 350 .newd:BL gbuff:BL fiddle:BL set
 370 .newl:B drvcol
 375 .drvcol2
 376  BL moun:SUBS R1,R1,#1
 390  SWI wr+3:SWI wr+6:SWI ws
 420  SWI wr+48:SWI wr+6:SWI ws
 441  SWI wr+31:SWI wr+33:SWI wr+6:SWI
ws
 442  EQUS"Dsc:":EQUB0:BL pxnam
 445  B prext
 446 .prext2
 555  BL tagst
 735  SWI ws:EQUS"          ":EQUB31
 736  EQUB 66:EQUB24:EQUB0
 740  LDR R0,[R1],#4:BL prdec
```

```
 775  BL ccli
1465 .pxnam
1466  MOV R0,#5:ADR R2,blok:SWI gb
1467  LDRB R3,[R2]:B libl
1740  BL prifx:BL tagst:B mloop
2500  EQUB0:SWI wr+3:MOV R15,R14
2631  EQUB19:EQUB7:EQUB16:EQUB255
2632  EQUB255:EQUB255
2633  EQUB19:EQUB2:EQUB16:EQUB255
2634  EQUB0:EQUB0
2635  EQUB19:EQUB0:EQUB16:EQUB0
2636  EQUB0:EQUB0
2651  EQUB19:EQUB3:EQUB16:EQUB0
2652  EQUB0:EQUB0
2653  EQUB19:EQUB4:EQUB16:EQUB0
2654  EQUB0:EQUB0
2655  EQUB19:EQUB5:EQUB16:EQUB128
2656  EQUB144:EQUB176
2730  EQUB17:EQUB3:EQUB31:EQUB0:EQUB24
2830  EQUB17:EQUB4:EQUB17:EQUB128+5
2846  EQUS"    (UN)TAG    "
2875 .gbuff
2876  ADR R2,buff:MOV R15,R14
3061 .drives
3062  SWI "ADFS_Drives"
3063  CMP R1,#1:CMPEQ R2,#0
3064  MOVEQ R15,R14
3065  MOV R0,#5:STRB R0,cob
3066  MOV R15,R14
3181 .tagsr
3182  MOV R0,#1:MOV R1,#0:ADR R2,tags
3183 .tarl
3184  STRB R0,[R2],#1:ADD R1,R1,#1
3185  CMP R1,R11:BNE tarl:MOV R15,R14
4021 .del12:BL tasq:BEQ delux
4022  LDRB R2,[R1,#&C]:TST R2,#8
4023  BEQ del2
4024  ADR R0,delms:SWI gen
4025 .delms
4026  EQUD00:EQUS"One or more files loc
ked":EQUB0
4027 .delux
4028  MOV R12,#0
4110  SWINE wr+ASC"R":TST R0,#2
4195  BL ccli
4280
4310  ADR R0,copq:ADD R0,R0,#5
4311  BL mbsb:SUB R0,R0,#1
4470  TST R1,#4:MOV R2,#ASC"R"
4481  MOV R2,#ASC"~":STRB R2,[R0],#1
4482  MOV R2,#ASC"Q":STRB R2,[R0],#1
4520  B go2
4521
4570  CMP R0,#&17:BEQ untg2
4695  BL ccli
5500  MOV R0,#&FF0:ADD R0,R0,#&F
5510  AND R0,R4,R0
5535  LDRB R6,[R5],#1:ADD R2,R2,R6,ASL#
8
5540  CMP R2,#&1000:BEQ ktyl1
5750  EQUW &FFF:EQUS"      ASCII"
5760  EQUW &FFE:EQUS"   Command"
5770  EQUW &FFD:EQUS"      Data"
5780  EQUW &FFC:EQUS"   PI Code"
5790  EQUW &FFB:EQUS"     BASIC"
5800  EQUW &FFA:EQUS"    Module"
5810  EQUW &FF9:EQUS"    Sprite"
5820  EQUW &FF8:EQUS" Abs. Code"
5830  EQUW &FF7:EQUS"  BBC font"
5840  EQUW &FF6:EQUS"Fancy font"
5845  EQUW &DDC:EQUS"Beebug ARC"
5850  EQUW &1000:EQUS"          "
5861 .untg2
5862  MOV R12,#0:BL tasq:BNE untag
5863  BL tagsr:B newd
5870 .fiddle
5875  STR R2,fpoi:STR R2,fbf
5880  MOV R0,#3:ADR R1,fvect:MOV R2,#0
5881  SWI "OS_Claim"
5884  ADR R0,fixf:SWI "XOS_CLI"
5885  LDR R2,fbf:STR R2,fpoi
5889  ADR R0,fixs:SWI "XOS_CLI"
5890  MOV R0,#3:ADR R1,fvect
5891  MOV R2,#0:SWI "OS_Release"
5892  MOV R0,#37:LDR R2,fbf
5893  LDRB R1,[R2,R0]:STRB R1,drive+1
5894  MOV R0,#0:ADR R3,title:MOV R4,#0
5895 .floop
5896  LDRB R1,[R2,R0]:STRB R1,[R3,R4]
5897  ADD R4,R4,#1:ADD R0,R0,#1
5898  CMP R4,#20:BNE floop
5899  MOV R0,#64
5900  ADR R3,urdir:MOV R4,#0
5901 .floop2
5902  LDRB R1,[R2,R0]:STRB R1,[R3,R4]
5903  ADD R4,R4,#1:ADD R0,R0,#1
5904  CMP R4,#13:BNE floop2
5910  MOV R15,R14
5911 .fvect
5912  STMFD R13!,{R2}
5913  LDR R2,fpoi:STRB R0,[R2],#1
5914  STR R2,fpoi:LDMFD R13!,{R2}
```

```
5915  LDMFD R13!,{R15}
5916 .fpoi EQUD 0
5917 .fbf EQUD 0
5920 .fixs:EQUS".":EQUB13
5925 .fixf:EQUS"FREE":EQUB13
5930 .drive:EQUS": ":EQUB0
5940 .title:EQUS STRING$(20,CHR$0)
5950 .urdir:EQUS STRING$(13,CHR$0)
5960 .prurd
5961  ADR R1,urdir:MOV R2,#10
5962 .prurdl
5963  LDRB R0,[R1],#1:SWI wc
5964  SUBS R2,R2,#1:BNE prurdl
5965  MOV R15,R14
5970 .prtit
5980  ADR R1,title:MOV R2,#9
5990 .prtitl
6000  LDRB R0,[R1],#1:SWI wc
6010  SUBS R2,R2,#1:BNE prtitl
6020  MOV R15,R14
6030 .prext
6040  SWI ws:EQUB31:EQUB63:EQUB6
6050  EQUS"Urd:":EQUB0
6060  BL prurd
6070  SWI ws:EQUB31:EQUB18:EQUB6
6080  EQUS"Tit:":EQUB0
6090  BL prtit
6100  SWI ws:EQUB31:EQUB48:EQUB7
6110  EQUS"Free:":EQUB0
6120  BL prfree
6125  SWI ws:EQUB31:EQUB63:EQUB7
6126  EQUS"Max:":EQUB0
6127  LDR R0,prmax:BL prdec
6150  B prext2
6160 .drvcol
6170  ADD R0,R1,#ASC"0"
6180  LDRB R2,drive+1
6190  CMP R2,R0
6200  SWI 256+17
6201  SWINE 256+4:SWIEQ 256+5
6202  SWI 256+17
6203  SWIEQ 256+132:SWINE 256+133
6210  B drvcol2
6220 .prmax
6230  EQUD 0
6240 .prfree
6241  STMFD R8!,{R14}
6242  ADR R0,drive
6243  LDRB R1,drive+1
6244  CMP R1,#ASC"0":BLT prfx
6245  CMP R1,#ASC"7":BGT prfx
6246  SWI "ADFS_FreeSpace"
6247 .prfx2
6248  STR R1,prmax:BL prdec
6249  LDMFD R8!,{R15}
6250  MOV R15,R14
6251 .prfx
6252  MOV R0,#0:MOV R1,#0:B prfx2
6260 .prdec
6265  STMFD R8!,{R0-R2,R14}
6270  ADR R1,xbuff:MOV R2,#10
6275  SWI "OS_BinaryToDecimal"
6280 .prdlp
6290  LDRB R0,[R1],#1:SWI wc
6300  SUBS R2,R2,#1:BNE prdlp
6310  LDMFD R8!,{R0-R2,R15}
6320 .xbuff:EQUS STRING$(20,CHR$0)
6330 .tagst
6340  STMFD R8!,{R0-R7,R9-R12,R14}
6350  MOV R12,#0:BL bloo:MOV R10,#0
6351  MOV R7,#0:MOV R6,#0:MOV R9,#0
6360  B tst1
6370 .tst0
6371  BL poin
6373  LDRB R0,[R1,#&10]:CMP R0,#2
6375  BEQ tst2:ADD R10,R10,#1
6376  LDR R0,[R1,#&8]:ADD R6,R6,R0
6380  BL tass:ADDEQ R9,R9,#1
6385  LDR R0,[R1,#&8]:ADDEQ R7,R7,R0
6386 .tst2
6390  ADD R12,R12,#1
6400 .tst1:CMP R12,R11:BNE tst0
6405  SWI ws:EQUB31:EQUB3:EQUB7
6406  EQUS"Tgf:              "
6407  EQUS"Tby:                ":EQUB0
6410  SWI ws:EQUB31:EQUB7:EQUB7:EQUB0
6420  MOV R0,R9:BL prdec
6430  SWI 256+ASC"/"
6440  MOV R0,R10:BL prdec
6450  SWI ws:EQUB31:EQUB22:EQUB7:EQUB0
6460  MOV R0,R7:BL prdec
6470  SWI 256+ASC"/"
6480  MOV R0,R6:BL prdec
6490  LDMFD R8!,{R0-R7,R9-R12,R15}
6500 .ccli
6505  SWI ws:EQUB31:EQUB3:EQUB6
6510  EQUS STRING$(74,CHR$32):EQUB0
6520  MOV R15,R14
7000 .end ]:ENDPROC
```

The complete version of the menu appears on this month's disc.

RU

A REAL-TIME IMAGE SPINNER (Continued from page 8)

```
 2540  CMP R0,#320<<16:ADDGE R15,R15,#4
 2550    CMP   R1, #256<<16:BLT   repeat
 2560 ]:=0
 2570 :
 2580 DEF FN_r_D(a)
 2590 [OPTZ:FN_init_loop:.repeat:FN_plot
 2600    SUB   R0, R0, R6:SUBS R1,R1,R7
 2610    ADDLT R15, R15, #4
 2620    CMP   R0, #0:BGE   repeat
 2630 ]:=0
 2640 :
 2650 DEF FN_init_resize
 2660 [OPT Z:STR R13,stack:STR R14,link
 2670    LDR R11,x_begin:LDR R12,y_begin
 2680    LDR   R6, x1:LDR   R7, y1
 2690    LDR   R14, x3:LDR   R13, output
 2700 ]:=0
 2710 :
 2720 DEF PROCrot(A%,B%,C%,D%,t)
 2730 xs=C%/1280:ys=D%/1024:z=1<<16
 2740 r=SQR(C%*C%+D%*D%)/8:B%=B%EOR1023
 2750 a=ATN(D%/C%):b=ATN(C%/D%):l=3.9E-3
 2760 IF 0<=t AND t<PI/2 THEN PROCset_up
1(t):CALL resize_1
 2770 IF PI/2<=t AND t<PI THEN PROCset_u
p2(t):CALL resize_2
 2780 IF PI<=t AND t<3*PI/2 THEN PROCset
_up1(t-PI):CALL resize_3
 2790 IF 3*PI/2<=t AND t<2*PI THEN PROCs
et_up2(t-PI):CALL resize_4
 2800 ENDPROC
 2810 :
 2820 DEF PROCset_up1(t)
 2830 IF SIN(t)*xs<l t=ASN(l/xs)
 2840 IF COS(t)*ys<l t=ACS(l/ys)
 2850 !x_begin=z*(A%/4-r*COS(a-t))
 2860 !y_begin=z*(B%/4+r*SIN(a-t))
 2870 !x1=z*COS(t)/xs:!y1=z*SIN(t)/ys
 2880 !y2 = z/COS(t)/ys:!x3 = z*TAN(t)
 2890 !x4=z/TAN(t):!x2=z/SIN(t)/xs
 2900 ENDPROC
 2910 :
 2920 DEF PROCset_up2(t)
 2930 IF SIN(t)*xs<l t=PI-ASN(l/xs)
 2940 IF COS(t)*ys>-l t=PI-ACS(l/ys)
 2950 !x_begin=z*(A%/4+r*SIN(b-t))
 2960 !y_begin=z*(B%/4+r*COS(b-t))
 2970 !x1=-z*COS(t)/xs:!y1=z*SIN(t)/ys
 2980 !x2=z/SIN(t)/xs:!x3=-z/TAN(t)
 2990 !y2 = -z/COS(t)/ys:!x4 = -z*TAN(t)
 3000 ENDPROC
```

RU

Pointer Definer

This classy pointer definer by Barry Christie really is a delight to use, with a highly professional screen display, and should prove most useful in developing your own mouse-driven programs.

The Archimedes is supplied with a mouse as standard, and most users rapidly become familiar with the default mouse-controlled screen pointer (an arrow). In fact, it is technically quite possible to design your own style of pointer, and use this in place of the original. Indeed, a Basic program may switch between four different pointer definitions under program control.

Defining your own pointer may seem a demanding task, but the utility listed here makes the whole process delightfully easy, and the resulting pointer definitions may be saved to disc, and then used as required in your own programs.

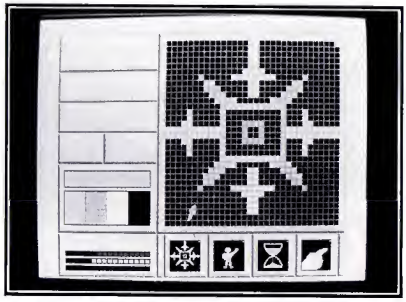
Carefully type in the Pointer Definer program and save safely to disc. When you run the Definer, you will see an enlarged grid area where the design of the pointer takes place. At the foot of the screen, up to four different pointer designs are shown at near to their proper size. A number of user-selectable options are displayed at the side of the screen.

DESIGNING A POINTER

A pointer may use up to three different colours, while black represents the *transparent* parts of the pointer design. Select the colour at any time from the colour display. You may also edit any colour selected, but not black, by using the red, blue and green slider bars at the bottom right of the screen. In this way, any of the Archimedes 4096 possible colours may be used.

In the main grid area, a pointer may be constructed by pressing the *select* button to colour in the small squares (representing pixels). The design will appear reduced in size at the foot of the screen. To select a different pointer, use the mouse to point to one of these four miniature displays. The program remembers up to four definitions at a time. The small flashing white square marks the *active* point for any pointer design, and the position of this may be changed using the *menu* button.

By selecting (with the mouse) appropriate options at the side of the screen display, the current (displayed) pointer definition may be saved as a disc file, or a previous definition reloaded (just press Return to abort a load or save operation). There is a.so the facility to temporarily replace the default pointer with your new creation (press the *adjust* button to return to normal editing), and an option to clear (to the current colour) the pointer edit area. Pressing Escape will exit from the pointer definer.



USING A POINTER

The saved pointer data provides the information needed by the system to define a new pointer. The way to use any such definition in your own programs is shown in the following short program (called MouseTest). You must reserve 400 bytes of memory for the initial loading of any pointer definition (line 60 loads a definition called *textdata* into the memory area labelled *mousedata*). Two values must then be poked into this data area (lines 70 and 80) to define the pointer number (range 0 to 3), and the address at which the data is located (in this case the label *mousedata*). Executing the SYS call at line 90 then defines the pointer (in the example as pointer 2), and this is activated by the MOUSE ON command at line 100. In use, the proportions of any pointer design, as with the default, depend on the mode used. Try modes 0, 1 and 2 in the demo program to see the effect of this.

```
   10 REM >MouseTest
   20 :
   30 MODE 0
   40 *POINTER
   50 DIM mousedata 400
   60 OSCLI("LOAD testdata "+STR$~(mouse
data))
   70 mousedata?&01=2
   80 mousedata!&06=mousedata+&0A
   90 SYS "OS_Word",&15,mousedata
  100 MOUSE ON 2
  110 END
```

A program may load any number of different pointer definitions, but no more than four can be defined at any one time. A program can switch pointers using the MOUSE ON <n> command to specify which is to be used. But, however many pointer definitions are loaded, only one 400 byte area of memory is required as a temporary storage area.

A number of pointer designs are included on the magazine disc with the programs.

```
   10 REM >PointerDEF
   20 REM Program   Pointer Definer
   30 REM Version   A 1.6
   40 REM Author    Barry W Christie
   50 REM RISC User October 1988
   60 REM Program   subject to copyright
   70 :
   80 MODE 12:OFF:ON ERROR MODE12:PRINT
REPORT$;" at line ";ERL:END
   90 PROCinitialise
  100 REPEAT
  110 MOUSE mxco,myco,mbut
  120 pxco=(mxco- 32) DIV 24
  130 pyco=(myco-224) DIV 24
  140 CASE TRUE OF
  150 WHEN FNpointer_area( 2,14,48,48,4)
:PROCpointer_point
  160 WHEN FNpointer_area( 2,14,48,48,2)
:PROCpointer_active
  170 WHEN FNpointer_area( 2, 2, 8, 8,4)
:PROCpointer_display(0)
  180 WHEN FNpointer_area(14, 2, 8, 8,4)
:PROCpointer_display(1)
  190 WHEN FNpointer_area(26, 2, 8, 8,4)
:PROCpointer_display(2)
  200 WHEN FNpointer_area(38, 2, 8, 8,4)
:PROCpointer_display(3)
  210 WHEN FNpointer_area(54,48,24, 4,4)
:PROCpointer_loaddata
  220 WHEN FNpointer_area(54,40,24, 4,4)
:PROCpointer_savedata
  230 WHEN FNpointer_area(54,32,11, 4,4)
:PROCpointer_show
  240 WHEN FNpointer_area(69,32, 9, 4,4)
:PROCpointer_clear
  250 WHEN FNpointer_area(54,14,24, 9,4)
:PROCpointer_colour
  260 WHEN FNpointer_area(54, 2,24, 6,4)
:PROCpointer_rgbedit
  270 ENDCASE
  280 UNTIL FALSE
  290 END
  300 :
  310 DEF PROCpointer_point
  320 pntr(p,pxco,pyco)=pcol
  330 IF pxco<>actp(p,0) OR pyco<>actp(p
,1) THEN PROCblock(pxco,pyco,pcol)
  340 ENDPROC
  350 :
  360 DEF PROCpointer_active
  370 x=actp(p,0):y=actp(p,1)
  380 PROCblock(x,y,pntr(p,x,y))
  390 actp(p,0)=pxco:actp(p,1)=pyco
  400 PROCblock(pxco,pyco,4)
  410 ENDPROC
  420 :
  430 DEF PROCpointer_display(pnew)
  440 IF pnew<>p THEN
  450 p=pnew
  460 FOR x=0 TO 31:FOR y=0 TO 31
  470 PROCblock(x,y,pntr(p,x,y))
  480 NEXT y:NEXT x
  490 PROCblock(actp(p,0),actp(p,1),4)
  500 PROCrgbset
  510 ENDIF
  520 ENDPROC
  530 :
  540 DEF PROCpointer_clear
  550 FOR pxco=0 TO 31:FOR pyco=0 TO 31
  560 PROCpointer_point
  570 NEXT pyco:NEXT pxco
  580 ENDPROC
  590 :
  600 DEF PROCpointer_loaddata
  610 IF FNfilename(0)<>"" THEN
  620 file=OPENIN(filename$)
  630 IF file=0 THEN
  640 VDU7:COLOUR 5
  650 PRINTTAB(54,VPOS-1)"No such file"
  660 t=INKEY(200):COLOUR 15
```

```
  670 PRINTTAB(54,VPOS-1)SPC24
  680 ELSE
  690 FOR b=1 TO 4:byte=BGET#file:NEXT
  700 actp(p,0)=BGET#file
  710 actp(p,1)=31-BGET#file
  720 FOR b=1 TO 4:byte=BGET#file:NEXT
  730 PROCgetprgbytes
  740 FOR data=0 TO 1023
  750 pntr(p,data MOD 32,31-(data DIV 32
))=data?mwrk
  760 NEXT data
  770 FOR colour=0 TO 2
  780 rgbp(p,colour,0)=BGET#file
  790 rgbp(p,colour,1)=BGET#file
  800 rgbp(p,colour,2)=BGET#file
  810 NEXT colour
  820 CLOSE#file
  830 p-=1:PROCpointer_display(p+1)
  840 ENDIF
  850 ENDIF
  860 ENDPROC
  870 :
  880 DEF PROCpointer_savedata
  890 IF FNfilename(1)<>"" THEN
  900 file=OPENOUT(filename$)
  910 BPUT#file,&00:BPUT#file,p+1
  920 BPUT#file,&08:BPUT#file,&20
  930 BPUT#file,actp(p,0)
  940 BPUT#file,31-actp(p,1)
  950 BPUT#file,&00:BPUT#file,&00
  960 BPUT#file,&00:BPUT#file,&00
  970 PROCgetsysbytes
  980 FOR data=0 TO 255
  990 BPUT#file,data?mwrk
 1000 NEXT data
 1010 FOR colour=0 TO 2
 1020 BPUT#file,rgbp(p,colour,0)
 1030 BPUT#file,rgbp(p,colour,1)
 1040 BPUT#file,rgbp(p,colour,2)
 1050 NEXT colour
 1060 CLOSE#file
 1070 ENDIF
 1080 ENDPROC
 1090 :
 1100 DEF PROCpointer_show
 1110 PROCdefine
 1120 FOR mcol=0 TO 2
 1130 MOUSE COLOUR mcol+1,rgbp(p,mcol,2)
,rgbp(p,mcol,1),rgbp(p,mcol,0)
 1140 NEXT mcol
 1150 MOUSE ON 2
 1160 REPEAT:MOUSE x,y,m:UNTIL m=0
 1170 REPEAT:MOUSE x,y,m:UNTIL m=1
 1180 PROCedit_pointer
 1190 ENDPROC
 1200 :
 1210 DEF PROCpointer_colour
 1220 pcol=(mxco-864) DIV 96:GCOL pcol
 1230 FOR s=0 TO 2:PROCrgbshow(s):NEXT
 1240 ENDPROC
 1250 :
 1260 DEF PROCpointer_rgbedit
 1270 IF pcol<>0 THEN
 1280 WHILE FNpointer_area(54,2,24,6,4)
 1290 rgbrgb=(myco- 32) DIV 32
 1300 rgbval=(mxco-864) DIV 24
 1310 rgbp(p,pcol-1,rgbrgb)=rgbval*16
 1320 PROCrgbset:PROCrgbshow(rgbrgb)
 1330 MOUSE mxco,myco,mbut
 1340 ENDWHILE
 1350 ENDIF
 1360 ENDPROC
 1370 :
 1380 DEF FNfilename(texty)
 1390 x=54:y=7+4*texty
 1400 PRINTTAB(x,y)"Filename  <"SPC12">"
 1410 VDU 7,31,66,y
 1420 SYS "OS_ReadLine",mwrk,10,32,126 T
O length;flags
 1430 filename$="":IF (flags AND 2)=0 TH
EN filename$=$mwrk
 1440 PRINTTAB(x,y)SPC24
 1450 =filename$
 1460 :
 1470 DEF PROCrgbset
 1480 FOR s1=0 TO 2:VDU 19,s1+1,16
 1490 FOR s2=0 TO 2:VDU rgbp(p,s1,2-s2)
 1500 NEXT s2:NEXT s1
 1510 FOR s=0 TO 2:PROCrgbshow(s):NEXT
 1520 ENDPROC
 1530 :
 1540 DEF PROCrgbshow(bar)
 1550 IF pcol=0 THEN rgblim=0 ELSE rgbli
m=rgbp(p,pcol-1,bar) DIV 16
 1560 rgbbar=32+bar*32
 1570 rgbwip=(15-rgblim)*24
 1580 GCOL 0:RECTANGLEFILL 1244-rgbwip,r
gbbar,rgbwip,19
 1590 GCOL 7-bar
 1600 FOR rgbbit=0 TO rgblim
 1610 RECTANGLEFILL 864+24*rgbbit,rgbbar
,19,15
 1620 NEXT rgbbit
 1630 GCOL pcol
 1640 RECTANGLEFILL 864,384,383,63
 1650 ENDPROC
```

```
 1660 :
 1670 DEF PROCblock(pntx,pnty,pntc)
 1680 GCOL pntc
 1690 RECTANGLEFILL 32+24*pntx,224+24*pn
ty,19,19
 1700 RECTANGLEFILL 68+192*p+4*pntx,32+4
*pnty,3,3
 1710 ENDPROC
 1720 :
 1730 DEF PROCdefine
 1740 mpar?&00=&00:mpar?&01=&02
 1750 mpar?&02=&08:mpar?&03=&20
 1760 mpar?&04=actp(p,0)
 1770 mpar?&05=31-actp(p,1)
 1780 mpar!&06=mpar+&0A
 1790 PROCgetsysbytes
 1800 FOR data=0 TO 255
 1810 ?(mpar+data+&0A)=data?mwrk
 1820 NEXT data
 1830 SYS "OS_Word",&15,mpar
 1840 ENDPROC
 1850 :
 1860 DEF PROCgetsysbytes
 1870 FOR dy=0 TO 31:FOR dx=0 TO 7
 1880 ?(mwrk+(31-dy)*&08+dx)=FNbuildbyte
 1890 NEXT dx:NEXT dy
 1900 ENDPROC
 1910 :
 1920 DEF FNbuildbyte
 1930 made=0:base=dx*4
 1940 FOR build=0 TO 3
 1950 bits=pntr(p,base+build,dy)
 1960 made=made OR bits<<(2*build)
 1970 NEXT build
 1980 =made
 1990 :
 2000 DEF PROCgetprgbytes
 2010 FOR data=0 TO 255
 2020 byte=BGET#file
 2030 FOR undo=0 TO 3
 2040 temp=byte AND (3<<(undo*2))
 2050 ?(mwrk+data*4+undo)=temp>>(2*undo)
 2060 NEXT undo:NEXT data
 2070 ENDPROC
 2080 :
 2090 DEF FNpointer_area(px,py,pw,ph,pb)
 2100 px=px*16:pw=px+pw*16-1
 2110 py=py*16:ph=py+ph*16-1
 2120 IF pb<>mbut THEN =FALSE
 2130 IF mxco<px OR mxco>pw THEN =FALSE
 2140 IF myco<py OR myco>ph THEN =FALSE
 2150 =TRUE
 2160 :
 2170 DEF PROCinitialise
 2180 DIM pntr(3,31,31),actp(3,1)
 2190 DIM rgbp(3,2,2),mpar 300,mwrk 1024
 2200 PROCedit_colours:OSCLI("POINTER")
 2210 PROCedit_pointer:PROCedit_screen
 2220 FOR s1=0 TO 3:FOR s2=0 TO 2
 2230 rgbp(s1,s2,2-s2)=255
 2240 NEXT s2:NEXT s1
 2250 pxco=0:pyco=31
 2260 FOR p=0 TO 3
 2270 PROCpointer_active
 2280 NEXT p
 2290 pcol=1:p=0:PROCrgbset
 2300 ENDPROC
 2310 :
 2320 DEF PROCedit_pointer
 2330 MOUSE COLOUR 1,192,192,192
 2340 MOUSE COLOUR 2,128,128,128
 2350 MOUSE COLOUR 3,0,0,0
 2360 MOUSE ON 1
 2370 ENDPROC
 2380 :
 2390 DEF PROCedit_colours
 2400 VDU 19,0,24,176,176,176
 2410 VDU 19,3,16,  0,  0,255
 2420 VDU 19,4,17,255,255,255
 2430 VDU 19,4,18,  0,  0,  0
 2440 VDU 19,5,16,255,  0,  0
 2450 VDU 19,6,16,  0,255,  0
 2460 VDU 19,7,16,  0,  0,255
 2470 VDU 19,9,16,176,176,176
 2480 FOR colours=10 TO 15
 2490 rgb=32*(colours-8)
 2500 VDU 19,colours,16,rgb,rgb,rgb
 2510 NEXT colours
 2520 OSCLI("FX  9,1"):COLOUR 137
 2530 OSCLI("FX 10,1"):COLOUR  15
 2540 ENDPROC
 2550 :
 2560 DEF PROCedit_screen
 2570 skip$=STRING$(4,CHR$9)
 2580 PROCbox( 0,12,52,52)
 2590 PROCbox(52,54,28,10)
 2600 PROCbox(52,46,28, 8)
 2610 PROCbox(52,38,28, 8)
 2620 PROCbox(52,30,15, 8)
 2630 PROCbox(67,30,13, 8)
 2640 PROCbox(52,12,28,18)
 2650 PROCbox( 0, 0, 2,12)
 2660 PROCbox( 2, 0,12,12)
 2670 PROCbox(14, 0,12,12)
 2680 PROCbox(26, 0,12,12)
```

Continued on page 30

ARE YOU GOOD ENOUGH?

As the leaders in software for the Archimedes range of computers, CLARES MICRO SUPPLIES are looking to extend our range even further. We are looking for people who are as excited by the Archimedes as we are.

If you have written any programs, completed or not, then we would like to hear from you.

If you have any ideas for programs and have the ability to execute the ideas then we want to hear from you.

If you have the ability to program the Archimedes but not the ideas to program then we want to hear from you.

Programs can be written in any language as long as they perform their stated task. Many of our programs contain large chunks of BASIC with ARM code in the areas that it is needed. BASIC on the Archimedes is a very powerful language and we do not attach any snob value to its use. If your program does what is meant to do then thats all we are interested in.

Why not join the top team on the Archimedes. You get the support of our in-house team, privileged access through us to Acorn and invitations to our informal programmers seminars.

The most important point is that you will be earning top royalty rates of if you prefer we will purchase your program outright.

Please write, in confidence, to Mr. D. Clare at:

Clares Micro Supplies,
98 Middlewich Road,
Northwich,
CHESHIRE CW9 7DA

If you have a program either complete or in development then please enclose a copy for our evaluation.

To protect yourself we advise that you lodge a copy of the program with your bank or solicitor BEFORE you send us a copy. You can then prove that your program pre-dates anything that we have.

Act today and become part of the leading software team producing software for the worlds fastest micro.

AutoSketch

Reviewed by Mike Williams.

Having recently reviewed Euclid by Ace Computing for RISC User (see Issue 8) I approached this review of AutoSketch with some enthusiasm. Graphics on the Archimedes can be very good indeed, so the opportunity to review another graphics package is not one to be missed.

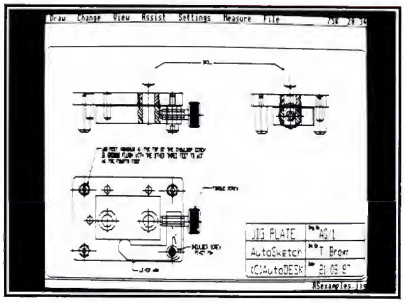
It must be pointed out immediately that AutoSketch serves quite a different purpose to Euclid. Whereas Euclid is concerned with the creation and manipulation of 3D objects on the screen, and in its approach is more of a design tool, AutoSketch is a 2D drafting package for producing orthogonal or isometric drawings or similar.

The very first impression of AutoSketch is highly encouraging. The custom packaging contains two 3.5" discs, a manual, an *Archimedes Installation Guide*, a *Quick Reference Guide*, and a booklet entitled *Getting the most out of AutoSketch*. All the printed items exhibit the same striking cover.

The installation guide reveals the true nature of AutoSketch, which was originally developed for PCs and compatibles before being converted to the Archimedes. All the other printed materials are as produced for the original version of AutoSketch. However, this is clearly acknowledged in the installation guide, which provides a tutorial and two appendices to supplement those in the original, and covers the Archimedes specific elements in the menus (use of sprites for example). The main manual, some 216 pages long, suffers from the narrow page width used, making the manual difficult to open fully.

Once installed, and configured for your system, the AutoSketch drawing screen appears on pressing Shift-Break. The display is pale grey, with a single line at the head of the screen for seven pull-down menus, and a further line at the foot of the screen for prompts, messages and user keyboard input. The seven menus are entitled:

```
Draw      Change     View       Assist
Settings  Measure    File
```



CREATING DRAWINGS

The *Draw* menu allows the user to select any of 9 'shapes' which can be drawn on the screen. These include points, lines, rectangles, circles, arcs and polygons, all as outlines only - no solid (or filled) shapes are possible. The Draw menu also allows text to be entered and positioned on the screen. One interesting inclusion is the curve option which uses splines to produce a smooth curve defined by a framework of points specified by the user. Previously saved drawings (or components) may also be imported to the drawing area.

Many of these Draw options work in conjunction with the *Settings* menu. This allows various properties or parameters to be specified which then form the basis of other options. For example, it is an option in this menu which allows the user to specify the height, angle, width and obliqueness of any text entered. Changes to these characteristics then apply to any subsequent drawing, but existing features remain unchanged.

The Settings menu has some other useful options. A *grid* can be superimposed on the screen with the user specifying the vertical and horizontal spacing. A *snap* option, when switched on, ensures that all points 'snap' to the nearest grid point. This is very useful for accuracy and correct alignment. Drawings may also be constructed as a set of up to ten *layers*, with full control over which layers are visible at any time. Another option provides a choice of ten different styles of line (continuous, dashed,

dotted etc). AutoSketch uses the Archimedes' seven basic colours for drawing on the light grey background, and any may be chosen as the current drawing colour. There is no option to redefine any colours though.

Drawing can be controlled either by the mouse, or for greater accuracy by the keyboard input of co-ordinates. These are quite different to the usual graphics units, and the screen dimensions run from 0 to 12 horizontally, and from 0 to 9 vertically, but fractions of units may be used. Co-ordinates entered through the keyboard may be absolute or relative Cartesian co-ordinates, or polar co-ordinates. Describing the latter, the manual confusingly refers to 'bearings', which are normally measured clockwise from north, but in fact uses the standard mathematical convention of anti-clockwise rotation starting from 'east'.

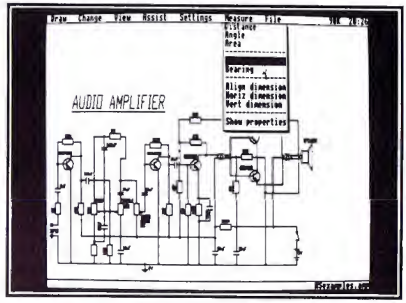
MAKING CHANGES

Editing your drawing requires the use of the *Change* menu, and this offers 13 different options in all. Many of the options involve *selecting* an element of the current display, and this is indicated on the screen by the replacement of the normal arrow pointer by a spread hand. Placing the hand over a line (or curve) for example will select that element. This is quite impressive, and AutoSketch clearly stores data on any drawing in a way which allows any individual object to be readily identified.

In practice, it can often be difficult to isolate an individual part of a drawing, so an alternative method of selection is provided. Pointing (with the hand) to a blank area of the screen allows a *window* or *crosses* box to be displayed. A window box selects anything that is totally enclosed by the box, while the crosses box selects anything that is even partly contained within the box. With practice, this proves both useful and easy. Furthermore, by enclosing a group of objects, the group can thenceforth be referred to as a single entity.

For erasing parts of a drawing the Change menu provides three options. *Erase* will delete from the screen any selected part of the drawing, even when that part overwrites others. If you erase anything by mistake, then the *Undo* option will literally undo the last action performed, including the use of Undo. If that begins to sound confusing, there is also a *Redo* function which restores the last step deleted by Undo.

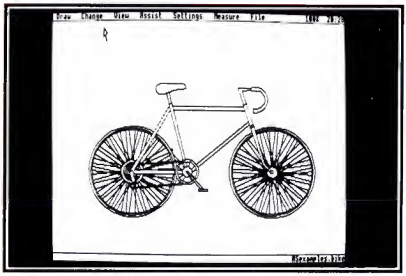
Other functions in the Change menu allow objects to be moved, copied, stretched, rotated, scaled or broken. Breaking an object divides it into two - making a break in a line for example. Some of these options, particularly stretching, are quite hard to follow from the manual and much trial and error becomes necessary before these features can be mastered.



OTHER MENUS

The View menu allows part of a drawing to be selected and enlarged, reduced or scanned to change the picture size and area. The whole picture can also be simply redrawn, very useful when multiple erases have left many small 'holes' in line work. The *Assist* menu simply toggles features, such as the current grid, off and on. There is also a *Measure* option which allows distances, areas and angles related of any part of a drawing to be calculated and displayed.

The last pull-down menu covers file handling, plotter information, and for the weary, and quite unexpectedly, an implementation of the game known as Connect Four. *Save* and *Open* options both make use of dialogue boxes to specify file names, and it is possible to change directories and even discs through this option. Drawings, or parts thereof, may also be saved as sprites, while a drawing can also be turned into a *slide* for subsequent display using the *SCREENLOAD command.



CONCLUSIONS

Like so many graphics packages on the Archimedes I find this one suffers from the medium resolution monitor used by Acorn with this system. A multi-sync monitor would make a considerable difference, and AutoSketch can be configured for mode 20 rather than its normal mode 12.

I would classify the package as workmanlike rather than outstanding, and there are many small features missing, such as the facility to draw lines with arrow heads for the insertion of dimensions. It would also have been useful to be able to cross-hatch shapes even if colour-filled shapes are not permissible. Other features might have included guide lines for two point perspective drawings.

The documentation tries very hard indeed, but even so does not always succeed, and some features, as I have mentioned, are quite difficult to follow. Overall, I believe that AutoSketch, as a drafting package for the Archimedes, is well worth considering at its price by those who require this type of application, but as with most applications ported or converted from other micros, I cannot help but feel that the Archimedes is capable of a good deal more.

Product	AutoSketch
Supplier	AutoDesk Ltd., 90 London Road, London SE1 6LN. Tel. 01-928 7868
Price	£91.54 inc VAT.

RU

ANIMATING ARCHIE (Part 3)

by Lee Calcraft

This month we look into the problems of animating more complex objects on the Archimedes.

Listings 1-3 SpriteSize 7 (2 on a 400 series).
Listing 3 ScreenSize 20 (5 on a 400 series).

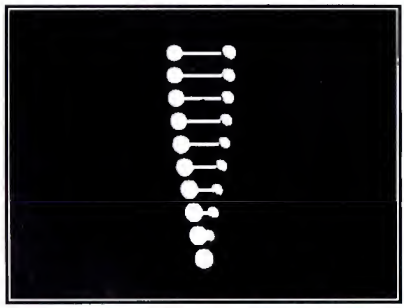
In the last issue we generated a sequence of images of a sphere, changing in size and angle of illumination, and created the impression of movement by displaying these in succession. We will now apply similar principles to a slightly more complex object.

Our object will be a two-atom molecule, consisting of a pair of spheres connected by a thin cylindrical bond. And we will again use the dithering routines from last month's Visuals to create the three component parts (i.e. the two atoms and the bond between them). Broadly speaking the approach adopted will be similar to that used last month: we will use a program to create a sequence of images, and save each to the sprite area after creation.

The program in listing 1 performs this task, generating a sequence of 31 sprites in mode 13. To run it you will need at least 50K of sprite space (use *Configure SpriteSize 7 on a 300 series machine, or 2 on a 400 series). When the program is run you will see each image created in the centre of the screen, and then copied to the bottom left, from where it is grabbed as a sprite. At the end of the program, the set of sprites is saved under the filename SgrMols31.

Listing 1

```
  10 REM              >3-1Anim
  20 REM Program      Create Mol Sprites
  30 REM Version      A 0.9
  40 REM Author       Lee Calcraft
  50 REM RISC User September 1988
  60 REM Program      Subject to Copyright
  70 :
  80 MODE13
  90 xpix%=4:ypix%=4
 100 horss=500
 110 molrad=100
 120 rads1=30:rads2=30
 130 light1=30:light2=30
 140 no=0
 150 :
```



Ten of the thirty one molecule sprites

```
 160 FOR phi%=0 TO 90 STEP 3
 170 phirad=PI*phi%/180
 180 cosphi=COS(phirad)
 190 sinphi=SIN(phirad)
 200 rad1=rads1-5*sinphi
 210 rad2=rads2+5*sinphi
 220 light1=-45-phi%:light2=phi%-90
 230 hors1=horss+molrad*cosphi
 240 hors2=horss-molrad*cosphi
 250 PROCsphere(hors1,500,rad1,light1,3
0,2)
 260 PROCcyl(hors2,500,hors1-hors2-cosp
hi*rad1+4,6,FALSE,30,6)
 270 PROCsphere(hors2,500,rad2,light2,3
0,8)
 280 MOVE 360,460:MOVE 640,540
 290 OSCLI("SGET "+STR$no)
 300 no+=1
 310 CLS
 320 PLOT &ED,0,0
 330 NEXT
 340 OSCLI("SSAVE SgrMols"+STR$(no))
 350 PRINTno;" sprites saved"
 360 END
 370 :
 380 :==================================
 390 DEFPROCcyl(X,Y,ht%,rad%,vert,L1%,c
ol%)
 400 ystep%=ypix%:xstep%=xpix%
 410 IF NOT vert THEN SWAP ystep%,xstep
%
 420 FOR Y%=0 TO ht% STEP ystep%
 430 A%=(rad%DIV xstep%)*xstep%
```

```
  440 FOR X%=-A% TO A% STEP xstep%
  450 P1%=DEG ASN(X%/rad%)
  460 D1=ABS(P1%-L1%)
  470 C%=7.99-D1/14-RND(1)
  480 IF C%<0 THEN C%=0
  490 GCOL0,col%+(C% AND 4)*5.25 TINT(C%
 AND 3)*64
  500 IF vert THEN PLOT69,X+X%,Y+Y% ELSE
 PLOT 69,X+Y%,Y+X%
  510 NEXT:NEXT:ENDPROC
  520 :=====================: ==========
  530 DEFPROCsphere(X,Y,rad%,L1%,L2%,col
%)
  540 FOR Y%=rad% TO -rad% STEP -ypix%
  550 A%=(SQR(rad%*rad%-Y%*Y%)DIV xpix%)
*xpix%
  560 FOR X%=-A% TO A% STEP xpix%
  570 P1%=DEG ASN(X%/rad%)
  580 P2%=DEG ASN(Y%/rad%)
  590 D1=ABS(P1%-L1%):D2=ABS(P2%-L2%)
  600 C%=7.99-SQR(D1*D1+D2*D2)/14-RND(1)
  610 IF C%<0 THEN C%=0
  620 GCOL0,col%+(C% AND 4)*5.25 TINT(C%
 AND 3)*64
  630 PLOT69,X+X%,Y+Y%:NEXT:NEXT
  640 ENDPROC
```

As you will notice, both the size of the spheres, and the lighting angle, are altered as the sequence proceeds. Both of these effects will add to the impression of movement when the molecule is finally displayed. If you take a look at the program, you will see that the loop counter, phi%, of the FOR loop which generates the image sequence, runs from 0 to 90 degrees in steps of 3 degrees. This is the angle between the X axis and the line between the two atoms - see Figure 1. All the variable parameters of the image, such as apparent length of cylindrical bond, apparent radius of the molecules, lighting angle, and so on, are dependent on this angle. To illustrate the principle, consider the horizontal position (hors1) of the right-hand molecule. This is calculated in line 230 by the expression:

```
hors1=horss+molrad*cosphi
```

where horss is the horizontal position of the centre of the molecule, molrad is half the distance between the centres of the two atoms, and cosphi is the cosine of the angle phi% in radians. The position of the other atom, and the apparent length of the cylindrical bond joining them is similarly dependent on phi%.

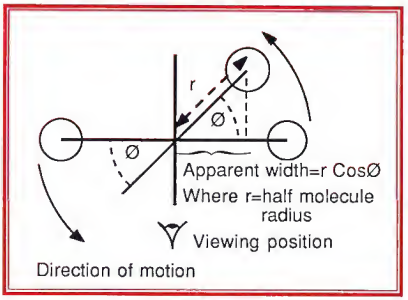


Figure 1

Once the program has been run, and the sprite file created, you can use listing 2 to animate the images. This short program causes the molecule to swing back and forth in a 90 degree arc as it bounces around the screen. By altering this program in various ways you can make the molecule move through any trajectory. You can also create quite different and complex objects by changing the program in listing 1 which generates the image sequence. But although you can make the sprite images as complex as you wish, there is a definite upper limit to the size of sprite used. If it is too large, the processor will still be plotting it when the VDU drivers access VDU RAM to update the monitor's screen, causing serious flicker. By using the WAIT statement we have linked sprite plotting to the frame scan rate, and so minimised the effect. But to free ourselves more fully from this limitation we can resort to a technique called screen flipping. This involves using dual screens, and drawing each sprite on the screen currently out of view, and then flipping screens to reveal the completed image.

Listing 2

```
  10 REM >3-2Anim
  20 REM Moves & rotates molecule
  30 REM Using SgrMols31 sprites
  40 :
  50 MODE13:OFF
  60 *SLOAD SgrMols31
```

```
 70 Y%=500:X%=500
 80 GCOL 3,0
 90 no=0:n=8:p=6:a=1
100 REPEAT
110 IF X%>1000 OR X%<100 THEN n=-n
120 IF Y%>800 OR Y%<100 THEN p=-p
130 X%+=n:Y%+=p
140 VDU23,27,0,no|
150 PLOT &ED,X%,Y%
160 WAIT
170 PLOT &ED,X%,Y%
180 no+=a:IF no=30 OR no=0 THEN a=-a
190 UNTIL FALSE
```

The program in listing 3, which requires a screen size of 160K, uses this principle. It is very similar to that in listing 2. The three most important additions are lines 120, 130 and 210. The first switches the value of b% between 1 and 2 on each cycle of the loop. The second uses the OSBYTE equivalent of *FX112 to determine which screen is written to. When b%=1 it is the normal screen, but with b%=2, the VDU drivers write to the shadow screen. Then in line 210 the displayed screen is also flipped. The sequence is as follows:

1. Display screen 1
2. Write to screen 2
3. Display screen 2
4. Write to screen 1

and so on.

This ensures that each sprite plotting operation is carried out on a screen hidden from view.

This saves time, because the observer can look at a given image for the whole time that it takes to create the next. You may remember that in our earlier program we could do very little while any given sprite was on screen, because we had to remove it before beginning to plot the next. Now we have so much time that we could make our sprites larger if we wished, and can take them right to the top of the screen without fear of flicker.

Listing 3

```
 10 REM >3-3Anim
 20 REM Moves & rotates molecule
 30 REM Using SgrMols31 sprites
 40 :
 50 ON ERROR MODE 12:REPORT:PRINT" at
line ";ERL:END
 60 MODE13:COLOUR 168
 70 *SLOAD SgrMols31
 80 Y%=500:X%=500
 90 no=0:n=8:p=6:a=1
100 b%=1
110 REPEAT
120 b%=1-(b%=1)
130 SYS "OS_Byte",112,b%:REM Driver
140 CLS
150 IF X%>1000 OR X%<0 THEN n=-n:SOUND
1,-15,70,2
160 IF Y%>950 OR Y%<0 THEN p=-p:SOUND
1,-15,70,2
170 X%+=n:Y%+=p
180 VDU23,27,0,no|
190 PLOT &ED,X%,Y%
200 WAIT
210 SYS "OS_Byte",113,b%:REM display
220 no+=a:IF no=30 OR no=0 THEN a=-a
230 UNTIL FALSE
```

In fact there is so much time between frames that you can plot an armada of swinging molecules with no loss of speed, by repeating the PLOT &ED statement with different co-ordinates (e.g. X%+100, Y%+100 etc.). You may also have noticed that we no longer need to set GCOL to 3 since we do not have to rely on Exclusive OR plotting to erase the previous sprite; we use CLS instead to clear the entire screen. As a result we can give our picture a coloured background (see line 60). To make this work properly you will need to re-run the sprite generator with a similarly coloured background (add COLOUR 168 to the end of line 80 in listing 1).

This flip-screen display technique eliminates some of the limitations associated with sprite plotting. But there remain two problems. The first is that using sprites eats up RAM very quickly if the objects are of any size, or if the sequence is to last more than a second or two. Secondly, you cannot very easily build up a complete scene because the background of each sprite will over-write anything in its path. This latter problem can be overcome by using transparency masks with each sprite, but this is a somewhat involved process. Next month we will introduce the Delta File technique which overcomes these problems in a highly economical manner.

RU

It's back!

The show that ALWAYS keeps you one step ahead

The premier exhibition for users of all Acorn machines returns to its popular venue in the heart of the capital.

Traditionally the liveliest event of the year on the Acorn calendar, the pre-Christmas show is the one you just cannot afford to miss.

It's your value-for-money passport to:

- 70 exhibitors displaying all the latest developments across the entire Acorn range.
- Archimedes World – which provides a fascinating glimpse into the current and future roles for this remarkable machine.
- Technical advice from the UK's leading experts on all Acorn computers.
- Hundreds of special offers for the BBC Micro and Electron waiting to be snapped up as top-value Christmas presents.

All this – and so much more – at the 20th record-breaking Electron & BBC Micro User Show.

You can even save yourself £1 before you get there by using this advanced ticket form.

New Horticultural Hall, Greycoat Street, London SW1

10am-6pm Friday, November 11
10am-6pm Saturday, November 12
10am-4pm Sunday, November 13

ELECTRON & BBC MICRO USER SHOW

Advance ticket order

SAVE £

Please supply tickets for November show:

☐ Adult tickets at £4 (save £1) ..... £.....

☐ Under-16s tickets at £2.50 (save £1) ..... £.....

☐ Cheque enclosed made payable to Database Publications Ltd. Total £ _____

☐ Please debit my credit card account: ☐ Access ☐ Visa Expiry date: / /

Name .....

Address .....

.....

.....Signed.....

Admission at door: £5 (adults) £3.50 (under 16s)
Advance ticket orders must be received by November 2, 1988

Post to: *Database Exhibitions, Europa House, Adlington Park, Adlington, Macclesfield SK10 4NP.*

DATABASE EXHIBITIONS

PHONE ORDERS: Ring Show Hotline: 0625 879920
Prestel Orders: KEY *89 THEN 614568383
MicroLink/Telecom Gold Orders: 72:MAG001
Please quote credit card number and full address

A496

THE MICRO USER INNOVATION AWARDS 1988

Take a stroll down Innovation Row – a brand new show feature area, specially constructed for the event.

See the grand finalists displaying their breakthroughs in public for the first time. And you can help pick the winners by casting a vote in both categories of the awards – BBC Micro and Archimedes.

How to get there

Underground: The nearest tube stations are VICTORIA (Victoria, District and Circle Lines), ST. JAMES'S PARK (District and Circle Lines) and PIMLICO (Victoria Line).

By British Rail: VICTORIA STATION. The halls are a 10-minute walk from the station.

By Bus: 11, 24, 29, 70, 76 and Red Arrow 507 to Victoria Street – alight Army and Navy Stores.

NEWS FROM THE PC SHOW

Mike Williams reports on some exciting news for Archimedes Users.

The PC show, held at Earls Court, London from the 14th to 18th September 1988, proved to be more than usually interesting for Acorn watchers. Several new developments, which may well have a profound effect on the future of the Archimedes, were unveiled to the public for the first time.

RISC OS FOR THE ARCHIMEDES

By far the most significant event was Acorn's latest development of Arthur for the Archimedes which offers a sophisticated multi-tasking environment. The new operating system, previous known as Arthur 2, has now been given the official name of RISC OS.

The main feature of the new OS is the incorporation of multi-tasking via the WIMP environment. Different tasks or applications may be active at the same time and can communicate with each other. The obvious and immediate beneficiary is the Desktop, which now provides a much enhanced level of user-friendliness, like the Apple Macintosh, but with extra colour.

Files may be 'dragged' from one location to another on the screen, and one application can be readily interrupted by another. The facility to build up 'layers' of activities provides a highly flexible and practical working environment.

RISC OS also includes three new bundled applications, a text editor, a new painting package and a drawing package. These application tools all look more than usually useful. Other utilities include an alarm clock, a calculator and an electronic mail system for network users. A new improved 6502 emulator, called 65Host, has also been incorporated.

The only bad news is that RISC OS will not be available until April 1989 according to Acorn, but the upgrade price (from Arthur 1.2 to RISC OS) is expected to be no more than £50.

ACORN DTP

Acorn was also demonstrating what is intended to be a flagship desktop publishing package for the Archimedes. This is based on the popular Timeworks package by GST, originators of the Arc's word processor, 1st Word Plus. Acorn DTP is fully window based, allowing page layouts to be created with ease. It appears to have most of the features required of desktop publishing, the only main weakness at present being the limited number of fonts available.

Acorn DTP is equally at home handling graphics images, which can be imported onto the page and rescaled and resized as required. This is where the multi-tasking comes to the fore. At any time the DTP package can be interrupted to use any other application to produce text or graphics for the DTP.

I was genuinely impressed by both RISC OS and Acorn DTP, and cannot wait to get my hands on both. I am only sorry that the majority of Archimedes users will have to wait till next year before they can do the same.

PROARTISAN FROM CLARES

If that were not enough, Dave Clares' demonstration of the ultimate in art packages for the Archimedes must have had many visitors to the show drooling at its stunning images.

The package uses 256 colours in mode 15, and these are displayed on-screen in four logically arranged colour charts. All the features of Clares' original Artisan remain, but have been significantly added to or improved. It is, perhaps, in the area of image distortion that the most amazing effects can be produced, with user defined sprites being distorted over the surfaces of cubes and spheres and many other shapes.

The colour fill function now has a novel feature, allowing the use of a range of shades from light to dark in any colour. Perhaps the most fascinating effect is that of the wash. Use the airbrush to spray a mosaic of coloured dots, and the wash gives an effect as though a few drops of water had been allowed to fall on the paper.

ProArtisan really is an amazing piece of software which we will be reviewing in full very soon. The only disappointment for most Archimedes users will be the price which is

£169.95, rather pricey even if it does include (for a limited period) a genuine wooden artist's palette box.

COMPUTER CONCEPTS TOO IMPULSIVE?

Computer Concepts was demonstrating an early version of its own DTP software, but enough to show that this has considerable potential. However, CC now appears to be having some second thoughts about its own Archimedes operating system, Impulse (see RISC User Issue 8). Maybe RISC OS is much better than CC anticipated. As a result, CC's own DTP package may well be written to run under RISC OS. However, work is still proceeding apace on Impulse, and we may even see two versions of their DTP software emerging, one for RISC OS and one for Impulse. Like everything else we shall have to wait and see.

CC also demonstrated a RISC card designed to turn any PC into a fully compatible Archimedes. Apparently CC has not yet decided whether this board should offer their own Impulse or Acorn's RISC OS as its operating system. More disappointingly, the promised fax add-ons for the Archimedes are still a long way off completion, and late 1988 or, (more likely) early 1989 was suggested for fully BABT approved systems.

DABS PRESS COMPILE A SURPRISE

Surprise new product from Dabs Press, better known for its popular books, was a working version of ABC, a Basic compiler for the Archimedes written by Paul Fellows, Acorn's ex–product manager for Arthur. The compiler is claimed to offer as much as a 40% increase in execution speed, and will be available very shortly for under £100.

For the Acorn market, the 1988 PC Show turned out to be one of the most interesting and exciting for years, and I probably haven't done justice to all the Archimedes products on display. We shall certainly be doing all we can to keep everyone fully informed of future developments. Indeed, we expect to cover RISC OS in our very next issue.

RU

MOUSE CONTROLLED CURSOR

This short module from Frank Wessels lets you replace the cursor keys (and others) by the mouse for virtually any application.

While many Archimedes programs are written to use the mouse, software such as the Basic Editor relies on the cursor keys to move around the text. The program listed here creates a relocatable module that makes moving the mouse simulate the cursor keys being pressed, and also lets the mouse buttons produce ASCII codes. Because of the way the module is written, the mouse can be used in almost any case where cursor keys would otherwise be needed. To use the program enter the listing, save it, and run it. This assembles the module and saves it to disc with the name 'CursorRM'. To load this module type *CursorRM. If this results in the error 'No room in RMA', type QUIT and try loading the module again.

Once the module is installed, it is controlled by a single command, *MouseCursor. Typing *MouseCursor On will cause the mouse's movements to move the screen cursor, while *MouseCursor Off will disable this. The command *MouseCursor On can be followed by up to three numeric parameters. These numbers specify the codes to be inserted into the keyboard buffer when the mouse buttons are pressed. The first value is for the left-hand button, the second for the middle one, and the third for the right-hand one. These numbers may be in decimal, or in hex if preceded by an '&'. For simple characters, the number is just the ASCII code of the character, while the codes for special keys are given on pages 166 and 168 of the *Programmer's Reference Manual*. We will be covering the subject of key codes in a future issue of RISC User. For example,

```
*MouseCursor On &8B &D &CD
```

will make the left-hand button mimic Copy, the middle button Return, and the right-hand button the Insert key.

```
   10 REM             >MouseCur
   20 REM Program     Mouse Cursor
   30 REM Version     A 1.0
   40 REM Author      Frank Wessels
   50 REM RISC User October 1988
   60 REM Program     Subject to copyright
   70 :
   80 DIM code &1000:SVCMode=3
   90 FOR pass%=4 TO 7 STEP 3
  100 O%=code:P%=0
  110 [ OPT pass%
  120 EQUD 0:EQUD 0:EQUD 0:EQUD 0
  130 EQUD title:EQUD help:EQUD command
  140 .title EQUS "MouseCursor":EQUB 0
  150 ALIGN
  160 .help EQUS "Mouse Cursor"+CHR$9+"1
.00 (01 Sep 1988)":EQUB 0:ALIGN
  170 .command EQUS "MouseCursor":EQUB 0
  180 ALIGN:EQUD mcursor:EQUD &00040001
  190 EQUD mcsy:EQUD mchelp:EQUD 0
  200 .mchelp EQUS "Use *MouseCursor On
[<n1>] [<n2>] [<n3>] to move the cursor
with the mouse.":EQUB 10:EQUB 13
  210 EQUS "Left, middle and right butto
n generate ASCII codes n1,n2 and n3 resp
ectively.":EQUB 10:EQUB 10:EQUB 13
  220 EQUS "Use *MouseCursor Off to disa
ble cursor movement with the mouse.":EQU
B 0:ALIGN
  230 .mcursor STMFD R13!,{R14}:LDRB R2,
[R0],#1:AND R2,R2,#&DF
  240 CMP R2,#ASC"O":BNE merr
  250 LDRB R2,[R0],#1:AND R2,R2,#&DF:CMP
 R2,#ASC"N":BEQ mcon
  260 CMP R2,#ASC"F":BEQ mcoff:BNE merr
  270 .mcon ADR R5,rb:MOV R4,#0:STRB R4,
rb:STRB R4,rb+1:STRB R4,rb+2
  280 SUB R3,R1,#1:MOV R4,R0:MOV R6,#2
  290 .gkcloop:SUBS R3,R3,#1:BMI gkccont
:BL getkeycode:BVS merr
  300 STRB R2,[R5,R6]:ADD R4,R1,#1:SUB R
6,R6,#1:B gkcloop
  310 .gkccont ADR R1,mblock:MOV R0,#21:
SWI "OS_Word":MOV R0,#&1C:ADR R1,tv
  320 MOV R2,#0:SWI "OS_Claim":LDMFD R13
!,{PC}
  330 :
  340 .merr ADR R0,error:MOV R1,PC
  350 ORR R1,R1,#1<<28:TEQP R1,#0
  360 LDMFD R13!,{PC}
  370 :
  380 .error EQUD 254:EQUS "Bad command
!":EQUB 10:EQUB 10:EQUB 13
  390 .mcsy EQUS "Syntax: *MouseCursor O
n [<n1>] [<n2>] [<n3>]":EQUB 10:EQUB 13
  400 EQUS "ASCII codes n1,n2,n3 given t
o left, middle and right button respecti
vely":EQUB 10:EQUB 13
  410 EQUS "Syntax: *MouseCursor Off":EQ
UB 10:EQUB 13:EQUB 0:ALIGN
  420 :
  430 .mcoff MOV R0,#&1C:ADR R1,tv
  440 MOV R2,#0:SWI "OS_Release"
  450 LDMFD R13!,{PC}
  460 :
  470 .getkeycode STMFD R13!,{R14}
  480 MOV R0,#&A000000A:MOV R1,R4
  490 MOV R2,#255:SWI "OS_ReadUnsigned"
  500 LDMFD R13!,{PC}
  510 :
  520 .tv STMFD R13!,{R0-R12,R14}
  530 LDRB R0,toggle:SUBS R0,R0,#1
```

Continued on page 30

Archimedes Visuals

This month's Visuals comprises a full-implementation of John Conway's Life, and a giant beach ball.

Mouse Driven Life

by Jim Walpole

In just 80 lines of code, this program achieves a full implementation of a mouse-driven Life. The start pattern is entered with the mouse, in a similar way to any drawing program (select to draw, adjust to erase). Pressing the menu button sets the display into motion. The program, which runs at several generations per second, can be interrupted at any point (with the menu button), and altered with the mouse, before re-running. The program uses bank switching to give a stable display, and makes use of fully legal point plotting routines, thus enabling it to work in any mode.

Buttons	Effect
Select	Draw
Menu	Start/stop display
Adjust	Erase
Adjust+Menu	Clear display

Controls

```
   10 REM             >Life
   20 REM Program     Mouse-driven Life
   30 REM Version     A 0.3
   40 REM Author      Jim Walpole
   50 REM RISC User October 1988
   60 REM Program     Subject to Copyright
   70 :
   80 ON ERROR MODE 1:REPORT:PRINT" at l
ine ";ERL:END
   90 DIM code% 25000:MODE 1:OFF
  100 FOR I%=0 TO 25000 STEP 4:code%!I%=
0:NEXT
  110 PROCassemble
  120 REPEAT:PROCsetup
  130 REPEAT:PROCgenerate:MOUSE X%,Y%,B%
  140 UNTIL B%=2:REPEAT:MOUSE X%,Y%,B%
  150 UNTIL B%=0:UNTIL FALSE
  160 :
  170 DEF PROCsetup
  180 screen=1:PROCwritescreen(screen)
  190 PROCshowscreen(screen)
  200 MOUSE ON:MOUSE TO 640,512
  210 REPEAT
```

```
  220 REPEAT MOUSE X%,Y%,B%:UNTIL B%
  230 IF B%=4 THEN X%=X%>>3:Y%=Y%>>3:fie
ld?(X%+Y%*160)=10:POINT X%<<3,Y%<<3
  240 IF B%=1 THEN X%=X%>>3:Y%=Y%>>3:fie
ld?(X%+Y%*160)=0:GCOL 0:POINT X%<<3,Y%<<
3:GCOL 3
  250 IF B%=3 THEN CLS:FOR I%=field TO c
ode%+25000 STEP 4:!I%=0:NEXT
  260 UNTIL B%=2:MOUSE OFF
  270 REPEAT:MOUSE X%,Y%,B%:UNTIL B%=0
  280 ENDPROC
  290 :
  300 DEF PROCgenerate
  310 CALL generate
  320 PROCwritescreen(3-screen)
  330 CLS:CALL update
  340 PROCshowscreen(3-screen):screen=3-
screen
  350 ENDPROC
  360 :
  370 DEF PROCwritescreen(n)
  380 SYS "OS_Byte",&70,n
  390 ENDPROC
  400 :
  410 DEF PROCshowscreen(n)
  420 SYS "OS_Byte",&71,n
  430 ENDPROC
  440 :
  450 DEF PROCassemble
  460 FOR opt%=0 TO 2 STEP 2
  470 P%=code%:[OPT opt%
  480 .generate
```

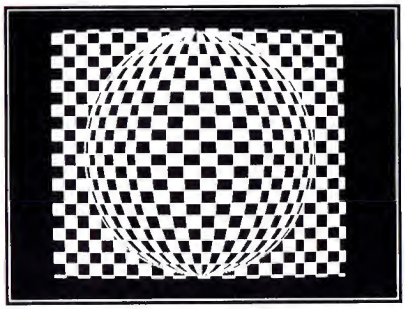
```
490 ADR R0,field:MOV R2,#0
500 .nextCellGen
510 LDRB R1,[R0,#+160]:CMP R1,#9
520 BLE skipGenerate
530 LDRB R1,[R0,#-1]:ADD R1,R1,#1
540 STRB R1,[R0,#-1]:LDRB R1,[R0,#-0]
550 ADD R1,R1,#1:STRB R1,[R0,#-0]
560 LDRB R1,[R0,#+1]:ADD R1,R1,#1
570 STRB R1,[R0,#+1]
580 LDRB R1,[R0,#+159]:ADD R1,R1,#1
590 STRB R1,[R0,#+159]
600 LDRB R1,[R0,#+161]:ADD R1,R1,#1
610 STRB R1,[R0,#+161]
620 LDRB R1,[R0,#+319]:ADD R1,R1,#1
630 STRB R1,[R0,#+319]
640 LDRB R1,[R0,#+320]:ADD R1,R1,#1
650 STRB R1,[R0,#+320]
660 LDRB R1,[R0,#+321]:ADD R1,R1,#1
670 STRB R1,[R0,#+321]
680 .skipGenerate
690 ADD R0,R0,#1:ADD R2,R2,#1
700 CMP R2,#20480:BNE nextCellGen
710 MOV R15,R14
720 .update
730 ADR R3,field:MOV R0,#69
740 MOV R1,#0:MOV R2,#0
750 .nextCellUp
760 LDRB R4,[R3,#+160]:CMP R4,#3
770 CMPNE R4,#12:CMPNE R4,#13
780 MOVEQ R4,#10:MOVNE R4,#0
790 SWIEQ "OS_Plot"
800 STRB R4,[R3,#+160]:ADD R3,R3,#1
810 ADD R1,R1,#8:CMP R1,#1280
820 MOVEQ R1,#0:ADDEQ R2,R2,#8
830 CMP R2,#1024:BNE nextCellUp
840 MOV R15,R14
850 EQUD 0
860 .field
870 ]NEXT:ENDPROC
```

Beach Ball

by Simon Proven

This very short routine generates a giant moving chequered beach ball. It is animated by repeatedly switching physical colour assignments. Note the use of the WAIT statement in line 290. This keeps flicker to a minimum by synchronising screen updating with the monitor's vertical flyback.

```
 10 REM            >Ball
 20 REM Program    Beach Ball
 30 REM Version    A 0.2
 40 REM Author     Simon Proven
 50 REM RISC User  October 1988
 60 REM Program    Subject to Copyright
 70 :
 80 MODE 12
 90 FOR N=0 TO 1280 STEP 1280/96
100 GCOL 15-(N DIV(1280/96)MOD 8)
110 RECTANGLE FILL N,0,1280/96,1023
120 NEXT
```



```
130 FOR N=0 TO 90 STEP 90/32
140 GCOL N DIV (90/32)MOD 8
150 VDU24,0;0;640;1023;
160 ELLIPSE FILL 640,512,COS(RAD N)*50
0,500
170 VDU24,640;0;1279;1023;
180 GCOL 7-(N DIV (90/32)MOD 8)
190 ELLIPSE FILL 640,512,COS(RAD N)*50
0,500
200 NEXT
210 VDU26:GCOL 3,4
220 FOR N=12 TO 1012 STEP 100
230 RECTANGLE FILL 0,N,1279,50
240 NEXT
250 REPEAT
260 FOR N=0 TO 7
270 COLOUR (N+4)MOD 8,1:COLOUR 8+(N+4)
MOD 8,4
280 COLOUR N,7:COLOUR N+8,7
290 WAIT:NEXT
300 UNTIL FALSE
```

RU

```
 2690 PROCbox(38, 0,12,12)
 2700 PROCbox(50, 0, 2,12)
 2710 PROCbox(52, 0,28,12):GCOL 0
 2720 FOR pointers=0 TO 3
 2730 RECTANGLEFILL 64+pointers*192,28,1
35,135
 2740 NEXT pointers
 2750 FOR rgb=0 TO 2
 2760 RECTANGLEFILL 860,28+rgb*32,387,23
 2770 NEXT rgb
 2780 RECTANGLEFILL 860,224,391,143
 2790 RECTANGLEFILL 860,380,391, 71
 2800 RECTANGLEFILL  28,220,771,771
 2810 GCOL 11
 2820 FOR grid=0 TO 32
 2830 RECTANGLE 28+grid*24,220,3,771
 2840 RECTANGLE 28,220+grid*24,771,3
 2850 NEXT grid
 2860 FOR gcol=0 TO 3
 2870 GCOL gcol:RECTANGLEFILL 866+16*(gc
ol*6),228,93,135
 2880 NEXT gcol
 2890 FOR t=1 TO 8
 2900 READ y,t$:PRINTTAB(54,y)t$
 2910 IF y=14 OR y=15 THEN READ t$:PRINT
TAB(69,y)t$
 2920 NEXT t
 2930 DATA 1,"THE MOUSE POINTER EDITOR"
 2940 DATA 2,"  By  Barry W Christie  "
 2950 DATA 3,"RISC User October 1988"
 2960 DATA 6,"Load previous definition"
 2970 DATA 10,"Save current  definition"
 2980 DATA 14,"Display the","Clear the"
 2990 DATA 15,"new pointer"," pointer "
 3000 DATA 27,"RGB  Content  Of  Colour"
 3010 ENDPROC
 3020 :
 3030 DEF PROCbox(x,y,w,h)
 3040 x=x*16:w=w*16-1:y=y*16:h=h*16-1
 3050 FOR boxtint=0 TO 3
 3060 GCOL boxtint+12
 3070 RECTANGLEFILL x+0,y,w-0,h-0
 3080 GCOL boxtint+10
 3090 RECTANGLEFILL x+4,y,w-4,h-4
 3100 x+=4:y+=4:w-=8:h-=8
 3110 NEXT boxtint
 3120 GCOL 9:RECTANGLEFILL x,y,w,h
 3130 ENDPROC
```

RU

MOUSE CONTROLLED CURSOR (Continued from page 27)

```
  540 MOVMI R0,#5:STRB R0,toggle
  550 BPL exit:MOV R9,PC
  560 ORR R8,R9, #SVCMode:TEQP R8,#0
  570 MOVNV R0,R0:STMFD R13!,{R14}
  580 SWI "OS_Mouse":LDMFD R13!,{R14}
  590 TEQP R9,#0:MOVNV R0,R0:MOV R8,R2
  600 ADR R6,rb:BL button:MOV R2,R5
  610 MOV R8,R8,ASR#1:ADD R6,R6,#1
  620 BL button:MOV R3,R5
  630 MOV R8,R8,ASR#1:ADD R6,R6,#1
  640 BL button:MOV R4,R5:MOV R7,#141
  650 SUBS R0,R0,#640:RSBMI R0,R0,#0
  660 MOVMI R7,#140:MOV R8,R0,ASR#4
  670 CMP R8,#0:MOVEQ R7,#0
  680 MOV R5,#143:SUBS R1,R1,#512
  690 RSBMI R1,R1,#0:MOVMI R5,#142
  700 MOV R6,R1,ASR#4:CMP R6,#0
  710 MOVEQ R5,#0:MOV R10,PC
  720 ORR R9,R10, #SVCMode
  730 TEQP R9,#0:MOVNV R0,R0
  740 STMFD R13!,{R14}:MOV R0,#138
  750 MOV R1,#0:CMP R2,#0:SWINE "OS_Byte
":MOV R1,#0:MOVS R2,R3:SWINE "OS_Byte"
  760 MOV R1,#0:MOVS R2,R4:SWINE "OS_Byt
e":MOV R9,#0:MOVS R2,R5:BEQ noymove
  770 .yloop STMFD R13!,{R0-R2}
  780 SWI "OS_Byte":LDMFD R13!,{R0-R2}
  790 SUBS R6,R6,#1:BNE yloop
  800 ADD R9,R9,#1
  810 .noymove MOVS R2,R7:BEQ noxmove
  820 .xloop STMFD R13!,{R0-R2}
  830 SWI "OS_Byte":LDMFD R13!,{R0-R2}
  840 SUBS R8,R8,#1:BNE xloop
  850 ADD R9,R9,#1
  860 .noxmove CMP R9,#0
  870 BEQ nomousemove:ADR R1,mblock
  880 MOV R0,#21:SWI "OS_Word"
  890 .nomousemove LDMFD R13!,{R14}
  900 TEQP R10,#0:MOVNV R0,R0
  910 .exit LDMFD R13!,{R0-R12,PC}
  920 :
  930 .button MOV R5,#0:MOV R7,#4
  940 TST R8,#1:BEQ bcont
  950 LDRB R7,[R6,#3]:CMP R7,#4
  960 LDREQB R5,[R6]:SUBS R7,R7,#1
  970 MOVMI R7,#0:LDRMIB R5,[R6]
  980 .bcont STRB R7,[R6,#3]:MOV PC,R14
  990 :
 1000 .mblock EQUB 3:EQUW 640:EQUW 512
 1010 .toggle EQUB 0:EQUW 0
 1020 .rb EQUB 0:.mb EQUB 0:.lb EQUB 0
 1030 .rbdelay EQUB 0:.mbdelay EQUB 0
 1040 .lbdelay EQUB 0
 1050 ]NEXT
 1060 SYS "OS_File",10,"CursorRM",&FFA,,
code,0%
```

RU

RISC USER TOOLBOX (Part 4)

David Spencer extends last month's disc sector editor, to enable named files to be examined.

The two commands added to the Toolbox last month both invoke the disc sector editor. The difference between them is in the way in which the start address is specified. The *DEDIT command specifies a particular start address on the disc, while *DEDITT allows the start to be specified in the form of head, track and sector numbers. This month we add a further command, *DEDITF, which takes a filename as its argument. The sector editor is then started at the first sector used to store the given file.

As before, the listing given below should be added to the complete program from last month. You must therefore ensure that the program from part 3 is not renumbered in any way. Once the new lines have been added, the program should be saved under a new name. Running the program will assemble the revised Toolbox and save it to disc. The Toolbox module can then be loaded as before. This month's magazine disc contains the source code for the complete Toolbox.

The syntax of the *DEDITF command is:

```
*DEDITF [<drive>] <pathname>
```

e.g.

```
*DEDITF 0 $.PROGRAMS.TOOLBOX
```

The pathname must not include a drive specification. If the optional drive number is not included, then the default drive (as set by *DRIVE) will be used. The sector editor will be started at the first sector used to store the named file, and can then be used as explained last month.

Because of the way that directories are stored on an ADFS disc, *DEDITF can also be used to examine a directory, simply by specifying the directory name in the command. However, it is not recommended that you actually *modify* a directory in this way, because although it is possible, it is all too easy to corrupt the directory's contents and lose access to all the files and sub-directories contained within it.

```
  329 EQUS "DeditF":EQUB 0
  330 ALIGN:EQUD deditfc:EQUD &20001
  331 EQUD defsyn:EQUD defhlp
 1401 .defhlp EQUS "*DeditF invokes the
disc editor at the start of the named fi
le.":EQUB 13
 1402 .defsyn EQUS "Syntax: DeditF [<dri
ve>] <pathname>":EQUB 0:ALIGN
 2657 EQUS "DiscFindAdd":EQUB 0
 3713 B swi7
 5895 MOV R6,#0:STR R6,[R12,#24]
 6285 LDR R4,[R12,#24]:ORR R4,R4,#1
 6286 STR R4,[R12,#24]
 6551 LDR R4,[R12,#24]:ORR R1,R1,R4,LSL
#2
 9601 STMFD R13!,{R0-R2}:MOV R0,#227
 9602 MOV R1,#&90:MOV R2,#0
 9603 SWI "OS_Byte":MOV R5,R1
 9604 LDMFD R13!,{R0-R2}:STMFD R13!,{R5}
 9831 LDMFD R13!,{R1}:MOV R0,#227
 9832 MOV R2,#0:SWI "OS_Byte"
10170 .deditfc STMFD R13!,{R14}
10180 LDR R12,[R12]:MOV R2,#2:BL gdrv
10190 STMFD R13!,{R0}:BL swi7
10200 LDMVSFD R13!,{R1,PC}
10210 MOV R3,R0:LDMFD R13!,{R0}
10220 BL swi4:B dedit2
10230 .adfs EQUS "ADFS"
10240 .rinfo STMFD R13!,{R1}
10250 LDR R1,[R12]:CMP R1,#0
10260 LDMEQFD R13!,{R1}:MOVEQ PC,R14
10270 STRB R0,[R12,R1]:ADD R1,R1,#1
10280 CMP R0,#ASC" ":MOVCC R1,#0
10290 STR R1,[R12]:LDMFD R13!,{R1}
10300 MOV PC,R14
10310 .swi7 STMFD R13!,{R1-R4,R14}
10320 ADD R2,R12,#1024:LDR R3,adfs
10330 STR R3,[R2],#4:MOV R3,#ASC":"
10340 STRB R3,[R2],#1:STRB R3,[R2],#1
10350 ORR R3,R0,#&30:STRB R3,[R2],#1
10360 MOV R3,#ASC".":STRB R3,[R2],#1
10370 .swi7_2 LDRB R3,[R1],#1
10380 CMP R3,#ASC" ":BEQ swi7_2
10390 .swi7_3 STRB R3,[R2],#1
10400 CMP R3,#ASC" ":LDRCSB R3,[R1],#1
10410 BCS swi7_3:MOV R0,#3
10420 MOV R1,#&74:SWI "OS_Byte"
10430 STMFD R13!,{R1}:MOV R0,#&1B
10440 ADR R1,rinfo:ADD R2,R12,#1280
10450 SWI "OS_Claim":MOV R0,#4
10460 STR R0,[R12,#1280]
10470 MOV R0,#9:ADD R1,R12,#1024
10480 SWI "XOS_FSControl"
10490 LDMFD R13!,{R1}
10500 STMFD R13!,{R0}:MOV R3,PC
10510 MOV R0,#3:SWI "OS_Byte"
10520 MOV R0,#&1B:ADR R1,rinfo
10530 ADD R2,R12,#1280:SWI "OS_Release"
10540 LDMFD R13!,{R0}:TEQP R3,#0
10550 LDMVSFD R13!,{R1-R4,PC}
10560 MOV R0,#16:ADD R1,R12,#1280
10570 ADD R1,R1,#59
10580 SWI "OS_ReadUnsigned"
10590 MOV R0,R2:LDMFD R13!,{R1-R4,PC}
```

RU

DYNAMIC BOXING

by Barry Christie and Lee Calcraft

Use this procedure to dynamically select any rectangle on screen by clicking the mouse.

The accompanying program allows the user to obtain the co-ordinates of any rectangular area of the screen. A small box of "moving dashes" similar to the WIMP window markers appears, and follows the pointer as the mouse is moved. The first click of the select button fixes the bottom left-hand corner of the rectangle. The pointer then jumps to the top right corner, allowing the size and proportions of the rectangle to be adjusted. Pressing select for a second time fixes the size, and causes the four parameters to be returned by the procedure. Alternatively, if menu is pressed, the pointer will flip back to the bottom left-hand corner, allowing the newly-sized rectangle to be moved around the screen and re-positioned.

Such a routine has many uses. In the example it is simply used for drawing randomly coloured boxes on the screen, while in the adjacent article it is used to grab sprites from any screen. If you are using the routine in your own programs, you will need all the code between the dotted lines. As you can see, the main procedure, PROCmarker, has four parameters, and each of these is returned after the procedure has finished its task. The parameters specify the co-ordinates of the bottom left-hand corner of the rectangle and its width and height, respectively. On entry, the first two parameters are set to the current position of the pointer, while the width and height are both set to 16 in the example.

We have also made use of the procedure PROCmousewait. This halts the program until a particular mouse button is pressed. The button number is given as a parameter, and if this is –1, the routine waits until any button is pressed. Used with the value zero, it is extremely useful for flushing the mouse.

```
  10 REM             >DynaBox
  20 REM Program     Dynamic Boxing
  30 REM Version     A 0.6
  40 REM Authors     Barry Christie
  50 REM             and Lee Calcraft
  60 REM RISC User October 1988
  70 REM Program     Subject to Copyright
  80 :
  90 MODE13
 100 *POINTER
 110 REPEAT
 120 PROCmousewait(0)
 130 x1=wx:y1=wy:x2=16:y2=16
 140 PROCmarker(x1,y1,x2,y2)
 150 GCOL 3,RND(63)
 160 IF mb<>1 THEN RECTANGLE FILL x1,y1
,x2,y2
 170 UNTIL FALSE
 180 :
1000 :==============================
1010 DEFPROCmarker(RETURN bx,RETURN by,
RETURN cx,RETURN cy)
1020 REPEAT
1030 PROCboxit(bx,by,cx,cy,1,2,4,TRUE)
1040 PROCmousewait(0)
1050 IF mb=4 THEN PROCboxit(bx,by,cx,cy
,1,2,4,FALSE)
1060 PROCmousewait(0)
1070 UNTIL mb=1 OR mb=4
1080 cx+=bx:cy+=by
1090 cx-=bx:cy-=by
1100 ENDPROC
1110 :
1120 DEFPROCboxit(RETURN mx,RETURN my,R
ETURN dx,RETURN dy,but1,but2,but3,base)
1130 MOUSE TO mx+(base+1)*dx,my+(base+1
)*dy
1140 dotstyle=0:*FX 21,9
1150 GCOL 3,7-(MODE=13 OR MODE=15)*56
1160 PROCmsquare
1170 REPEAT
1180 PROCmsquare:MOUSE tx,ty,mb
1190 IF base THEN mx=tx:my=ty ELSE dx=t
x-mx:dy=ty-my
1200 dotstyle=(dotstyle+1) MOD 8
1210 PROCmsquare 1220 UNTIL mb=but1 OR
mb=but2 OR mb=but3
1230 PROCmsquare
1240 bx=mx:by=my:cx=dx:cy=dy
1250 ENDPROC
1260 :
1270 DEF PROCmsquare
1280 VDU 23,6,&FCFC>>dotstyle|
1290 PLOT  4,mx,my:WAIT
1300 PLOT 21,mx+dx,my
1310 PLOT 21,mx+dx,my+dy
1320 PLOT 21,mx,my+dy:PLOT 21,mx,my
1330 ENDPROC
1340 :
1350 DEFPROCmousewait(n)
1360 REPEAT:MOUSE wx,wy,wz
1370 UNTIL wz=n OR (n=TRUE AND wz>0)
1380 ENDPROC
1390 :==============================
```

RU

SPRITE GRABBER AND PAINTER

by Lee Calcraft

Add these lines to the Dynamic Boxing routine to produce a general-purpose sprite grabber and painter

This program requires some sprite space

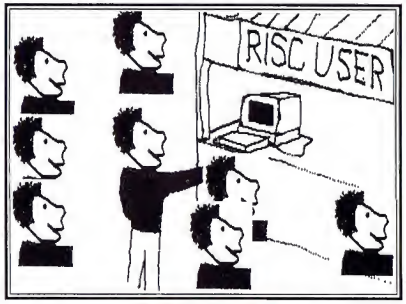
If you type in the lines below, and add to them the procedures between dotted lines from the Dynamic Boxing listing you will have a program capable of grabbing any parts of a screen as a series of sprites, and of painting anywhere on the screen using the last-grabbed sprite as a brush. To grab a sprite, press select twice (once for its position, once for its size, as described above). To paint with the last grabbed sprite, use menu. Use adjust to quit, and if you wish to save the sprites, use:

```
*SSAVE sprites
```

after quitting the program. Use *SLIST to catalogue the sprites in the sprite area.

Grabbing a Sprite

```
 10 REM              >SpriteG
 20 REM Program      Sprite Grab & Paint
 30 REM Version      A 0.3
 40 REM Author       Lee Calcraft
 50 REM RISC User    October 1988
 60 REM Program      Subject to Copyright
 70 :
 80 MODE13
 90 *SNEW
100 *SCREENLOAD SCREEN
110 name$="Sprite":no=0
120 *POINTER
130 REPEAT
140 PROCmousewait(-1)
150 CASE wz OF
```



Plotting Sprites

```
160 WHEN 2:GCOL 0,0:PLOT &ED,wx,wy
170 WHEN 4:PROCgrab
180 ENDCASE
190 UNTIL wz=1
200 END
210 :
220 DEFPROCgrab
230 x1=wx:y1=wy:x2=16:y2=16
240 PROCmarker(x1,y1,x2,y2)
250 IF mb<>1 THEN
260 MOVE x1,y1:MOVE x1+x2,y1+y2
270 OSCLI ("SGET "+name$+STR$(no))
```

Painting with Sprites

```
280 no+=1
290 ENDIF
300 ENDPROC
```

RU

SPEEDING UP BASIC

David Spencer and Mike Williams offer some advice on speeding up your Archimedes Basic programs.

Fast as the Archimedes Basic V is, especially when running RAMBASIC from the welcome disc, many programs written in Basic can benefit from the appropriate choice of Basic instructions. The purpose of this article is to examine alternative ways of programming in Basic in order to produce more efficient programs. To this end we have timed a number of commonly used features of the language. We must stress that we are only concerned with Basic programming techniques. For example, given a large list of names to sort into order, we are interested only in the best choice of Basic instructions for the implementation, and not in choosing which sort algorithm to use.

The first step when trying to speed up a program is to identify the time-critical areas of the program. These are the areas that will benefit most from a few speed 'tweaks'. There is in fact a rule of thumb called the 80-20 rule. This states that it is the often the case that approximately 80% of the time taken to execute a program is spent on just 20% of the code. Clearly, if you can identify and speed up this 20%, you gain much more than if you speed up the other 80%. As long as you understand the workings of a particular program, it should be possible to find the time-critical section. For example, this may be a loop that is executed many times, while the rest of the program only executes once.

PROGRAM LAYOUT

The first approach to speeding up any program is to ensure that all unnecessary elements are removed. REM statements, single colon lines and blank lines might not actually perform any function, but it does take Basic a finite time to interpret them, even on an Archimedes. Therefore, if you remove all such lines, the program will run slightly faster, albeit at the expense of clarity. Similarly, the Basic interpreter takes longer to find the next line, than it does to find the next statement after a colon. Therefore, using multi-statement lines wherever possible will also increase the speed.

LOOPS

Any code contained within a program is likely to be executed many times. Therefore, if the speed of the operations within a loop can be increased, the overall speed improvement will be many times greater. One obvious move is to ensure that the loop contains no unnecessary operations. For example, if there is an expression whose arguments will be unchanged on each iteration of the loop, then this need only be evaluated once before the loop starts, rather than on each iteration. Table 1 shows the times taken to execute an empty loop 100000 times using four different techniques. The first is a simple loop using IF-THEN GOTO, the second is a FOR-NEXT loop, the third uses REPEAT-UNTIL, and the final one uses WHILE-ENDWHILE.

GOTO	12.86 s
FOR NEXT	1.48 s
REPEAT UNTIL	12.18 s
WHILE ENDWHILE	12.26 s

Table 1.

The actual timings will depend on the operations performed within the loop, but the time overhead will remain the same. The loop using GOTO is the slowest because the interpreter has to search through the program for the appropriate line each time GOTO is executed. If our test loop was at the end of a long program, the GOTO loop would have been much slower than the others. For example, the same GOTO loop took 14.54 seconds when placed at the end of a 1000 line program. All the other times remain constant wherever the loop is in the program.

PROCEDURES AND SUBROUTINES

There are three ways of calling subroutines. These are GOSUB, PROC and FN. Table 2 shows timings for 10000 dummy calls using each technique. For PROC and FN, the timings are given for no parameters, for a single dummy parameter, and the same parameter but with RETURN used in the definition.

GOSUB	4.90 s
PROC	5.71 s
PROC(A)	15.31 s
PROC(RETURN A)	20.50 s
FN	12.00 s
FN(A)	21.52 s
FN(RETURN A)	26.90 s

Table 2.

As you can see, FN is slower than PROC, because a value must be returned, and the use of a RETURN parameter is slower than a normal parameter for the same reason. The addition of any form of parameter slows execution down. The GOSUB timing is deceptive, because as said earlier, this will increase as program length increases.

SWITCHES

A switch is a program structure in which one of several options is executed according to a condition. The simplest switches are ON-GOTO, and ON-GOSUB. Both of these should be avoided for the reasons given earlier. A more elegant solution is ON-PROC which is much faster than the previous two. However, if you use this method observe the warning given above regarding parameters. The CASE statement is provided specifically for performing switches, while another alternative is to use the multi-line form of IF-THEN-ELSE to nest a set of comparisons and actions. Using the four methods to implement a simple switch that takes a number from 0 to 9 and prints out "ZERO" to "NINE", gives the timings for 10000 runs given in Table 3.

ON-GOSUB	10.96 s
ON-PROC	11.36 s
CASE	11.95 s
IF-THEN	14.57 s

Table 3.

The first timing is deceptive, because when the subroutines are placed at the end of a 1000 line program, the time increases to 12.16 s, while the others remain unchanged.

VARIABLES AND EXPRESSIONS

One of the best ways to speed up a program that performs lots of arithmetic is to use integer variables wherever possible. In particular, the resident integer variables, A%-Z%, are very fast. As a comparison, incrementing A% from zero 100000 times takes 8.20 s, while using a% as the variable takes 8.42 s, and using the real variable A takes 10.24 s. Obviously, where all calculations involve only integers, it is sufficient simply to ensure that all the variables used are integer variables. Incidentally, beware of using /. This always performs floating point division, even if both arguments are integers. The way around this is to use the DIV operator instead. The expression 1/1 evaluated 100000 times takes 10.02 s, while replacing the '/' with DIV reduces the time to 8.79 s. Another speed 'tweak' is to use TRUE and FALSE instead of -1 and 0. This offers a 10% speed increase.

Make sure that you use Basic V's new matrix instructions wherever possible. These are significantly faster than programming your own FOR-NEXT loops to handle each element separately. For example, incrementing each element of a 10 by 10 matrix, an element at a time, for a total of 10000 times takes 175 seconds, while doing the same incrementing with a whole array operation takes just 24 seconds! Similarly, the use of << and >> for shift operations is quicker than using multiplication and division.

When using string variables, even with Basic V, it is always best to initialise them to their maximum length at the start of the program. Otherwise, when the length of a string expands, Basic has to allocate more space, and this all takes time.

Variable names themselves should be kept as short as possible for extra speed. It is much quicker to locate a single letter variable name than it is to find a ten character name. Because of the way in which Basic stores variables, the names should also be distributed across the alphabet. For example, don't start all variable names with an 'A'.

CONCLUSION

Hopefully, the suggestions given above will enable you to squeeze the most out of your Basic V programs. However, as they say, 'You don't get owt for nowt', and coding a program for speed will quite frequently have an adverse effect on its length or legibility.

As a final word, it doesn't matter how much you try and speed up a badly written program, it will never be as efficient as it could be. You should, therefore, concentrate on developing efficient, fast, algorithms at the outset, and then code them to run quickly, as the potential benefits are much greater. RU

DABS PRESS

Dabhand User News

Alerion is the first traditional shoot-em-up game for the Archimedes brought to you by the experts: **David Atherton** and **Bruce Smith**. Written in ARM machine code it uses the spectacular speed, sound and colour of the Archimedes. We reckon that it's impossible to finish and will give you hours of addictive fun for just £14.95.

If you're a serious user then our 368 page Dabhand Guide on **Archimedes Assembly Language** is an absolute must and is packed full of information and advice on programming the Archie.

DON'T MISS our advert in next month's issue of Risc User for full details on a major new product for the Archimedes that is sure to change the way in which you write and run your BASIC programs.

If you can't wait until then just write or phone us for our new catalogue which details **SIX** new products for the Archimedes. And it's absolutely free of charge. **Act now!**

Alerion

Archie Arcade Action!

"A welcome return to the traditional shoot-em-up"

Alerion - an eagle without beak or feet is the Arcturian term for impossible and the codename for your mission in this exciting all action game.

Your space-fighter is equipped with revolutionary new equipment, not least a new radar cloaking system which renders you invisible, a holographic targeting system and unlimited fire power. To succeed your task is quite simple ... blow the living daylights out of anything that moves!

You have a bird's eye view of the action, your space-fighter flying over the varied enemy terrain, scrolling effortlessly beneath you. This is done by using the impressive 256 colour, high resolution mode. The game screen is refreshed 25 times a second by using two 80k screens in a highly innovative fashion, where one is dedicated to update while the other is used solely for display purposes. There is therefore no messy screen swapping. Just as impressive is the use of digital sounds which were sampled from professional sources giving the game an authentic feel.

Alerion is a welcome return to the most popular of all computer games but utilising the power speed, sound (not A305) and superb graphics only available on the fastest micro in the world.

Alerion costs £14.95 and is available now!

Archimedes Assembly Language: A Dabhand Guide

The first book specifically written for the Archimedes to provide a complete guide to programming the Archimedes in machine code. Whether you are an Archimedes owner, a user, or just interested in the ARM chip this book is the definitive guide.

In a massive 368 pages, author Mike Ginns provides a clear, step by step account of using the assembler. Many simple, useful, documented programs provide the practice to illustrate the theory making it ideal for the beginner.

Here's what Risc User said in their review: *"The style of the text throughout the book is easy to read. Good, clear diagrams are used to make explanations easier. I would recommend Archimedes Assembly Language..."* At just £14.95 this Dabhand Guide represents full value for money. A programs disc is available for £9.95 or the two may be purchased for £21.95 when ordered together.

C: A Dabhand Guide

Available from all good bookshops

This superb 512 page book is one of *the* publications of 1988 and provides a step-by-step introduction and programming guide to C on the Archimedes. Beebug had this to say: *"...the book being full of good advice about program design and layout...a very good, resonably priced introduction to C for the non-specialist."* Book £14.95. £21.95 with disc.

Free : New Catalogue and Ordering Details

Write to us, phone us, or send us a mailbox and we will send you, free of charge, our information packed catalogue giving full details of all our current and some of our forthcoming Archie products. We'll also send you details of new books and software.

Send cheques, POs, official orders to the address below, or quote your Access/Visa card number and expiry date. Credit card orders accepted by phone, letter or mailbox. P&P free in UK/BFPO. Elsewhere add £2.50 or £12 airmail.

Dabs Press (RU), 5 Victoria Lane, Whitefield, Manchester, M25 6AL. Phone 061-766-8423 (24 hours)
Prestel 942876210 • BT Gold 72:MAG11596

INTRODUCING ARM ASSEMBLER (6)

by Lee Calcraft

This Month: Branch Instructions, Subroutines and Stacks.

SIMPLE BRANCHING

The ARM has a very straightforward branch instruction which causes a program to branch to the address given. The instruction takes a single parameter, which may be any logical address in the ARM's memory map, or any expression (such as a label) which yields an address at assembly time. This address must of course be word-aligned. For example:

```
B loop
```

As with all ARM instructions, any of the 16 condition suffixes may be appended, yielding instruction mnemonics highly reminiscent of 6502 assembler:

```
BNE loop
BEQ loop
```

and so on.

There are two points worth noting in connection with the branch instruction. The first is that on assembly, the assembler always generates a position independent branch instruction, expressing the destination address relative to the PC. Secondly, each branch instruction carries a timing penalty because pipelining is lost. With the ARM's pipelining, remember, the processor simultaneously performs the current instruction while decoding the next, and fetching the one after. Clearly if a branch is executed, the results of the advance decode and fetch operations must be jettisoned. A branch instruction carried out in RAM on an 8MHz ARM takes 500 nanoseconds compared to 125 nanoseconds for an MOV, ADD or SUB instruction.

BRANCH WITH LINK

There is just one variant of the branch instruction: Branch with Link (BL). It is identical to the normal Branch instruction in every way, including timing, except that it places the contents of the program counter into register 14 immediately prior to branching. It can thus be used to implement subroutines, since the contents of R14 can be used as a return address. To return from the subroutine, all you need to do is to move the contents of R14 back into the program counter. The ARM itself handles all the problems caused by pipelining, and the programmer does not need to correct the return address in any way. Thus the following implements a short subroutine called beep:

```
CMP  R0,#&20
BLLO beep
;  Rest of code
.
.
.beep
SWI 256+7
MOV PC,R14
```

Note that the branch with link is used with the LO suffix so that the subroutine is only executed if R0 contains less than &20. The subroutine itself produces the beep by executing the equivalent of VDU7. It then moves R14 back into the program counter, and so returns to the main body of the program. If you wish to restore the flags to the condition prior to the subroutine call, you should use an "S" suffix in the return instruction:

```
MOVS PC,R14
```

otherwise the state of the flags on return will be those resulting from the subroutine itself. Of course in this simple example the flags play no part, and the routine does so little that it would have been much more efficient to have replaced the subroutine call with:

```
SWILO 256+7
```

Generally speaking, subroutines are used to make a program more modular, and therefore easier to follow, and also to avoid the duplication of frequently used pieces of code. But a Branch with Link (with accompanying return instruction) carries a 1 microsecond time overhead, and should be avoided in time-critical

parts of a program. One further application of Branch with Link is to be found in Basic's CALL statement and USR function. Both of these store a return address to Basic in R14 before executing the specified user-supplied machine code routine. This is why we use the instruction:

```
MOV PC,R14
```

to return to Basic at the end of each machine code program.

STACKING RETURN ADDRESSES

In practice, subroutines will play a major role in most machine code programs, and will, moreover, be nested where appropriate. In such cases, we cannot permanently store all return addresses in R14 - since each would overwrite the last. Indeed, even calling one subroutine would cause Basic's return address to be lost. We must therefore implement a stack, pushing return addresses onto the stack when a subroutine is called, and pulling them off when a return is made.

In keeping with its reduced instruction design philosophy the ARM has no dedicated stack, but provides instructions which allow the user to create his own. The key to this is the multiple load and store instructions introduced last month. An example will help here:

```
;Main program
STMFD R13!,{R14}
BL  subroutine
;Rest of program
.
.
LDMFD R13!,{R14}
MOV PC,R14        ;Return to Basic
.
.subroutine
;Subroutine code
MOV PC,R14
```

The only real difference between this and the previous example is that the main program begins by stacking the contents of R14, and restores this register immediately prior to returning to Basic. This is achieved by using variants of the multiple load and store instructions. The two suffixes, F and D determine that the stack should be Full, and Descending. This is the Acorn standard, and simply means that the stack pointer points to the last full location, and that the stack grows downwards in RAM. The other two possible suffixes are E (for empty) and I (for incrementing), but these options are very rarely used.

Incidentally, note the use of the '!' symbol after R13 in the stack instructions. This ensures that right-back occurs, and that register R13 is updated each time the stack is accessed. Because Basic uses the same '!' symbol as an indirection operator, the assembler will get confused if you use a variable name for R13 when specifying right-back. Thus:

```
LDMFD stack!,{R14}
```

will be mis-interpreted. To avoid this, use parentheses:

```
LDMFD (stack)!,{R14}
```

You will see that we have used register R13 as the stack pointer, and that we have not assigned a value to its contents. This is because Basic uses R13 as a stack pointer, and an area or RAM is thus already assigned by Basic for stack use. Before returning to Basic we must therefore always ensure that we leave the stack in exactly the same state as we found it.

The other implication of this is that you should not generally use register R13 for anything else, because when you return to Basic, if the contents have changed, then any stacked addresses, such as procedure return addresses, and so on, will be lost, and Basic will report an error.

In our example above, the last two instructions in the main program are:

```
LDMFD R13!,{R14}
```

```
MOV PC,R14
```

These could be combined to give the single instruction:

```
LDMFD R13!,{PC}
```

This simply restores the top address on the stack to the program counter, rather than restoring it to register 14, and then moving it into the program counter. If you want the flags to be restored in a multiple load instruction, you can add a final circumflex character, thus:

```
LDMFD R13!,{PC}^
```

This can be used with any variant of LDM in which the program counter is one of the destination registers.

NESTED SUBROUTINES

It may not have escaped your notice that the subroutine in our example above does not stack R14. This is perfectly ok providing that the subroutine does not itself call another. Where this occurs R14 should be stacked in exactly the same way as in the main routine. Thus:

```
;Main program
STMFD R13!,{R14}
.
BL firstsub
.
LDMFD R13!,{PC} ;Return to Basic
.
.
.firstsub
STMFD R13!,{R14}
BL secondsub
LDMFD R13!,{PC}  ;Return from 1st sub
.
.
.secondsub
;code
MOV PC,R14       ;Return from 2nd sub
```

If you need to add a third level of nesting, then the second subroutine will also need to stack R14. And there is something to be said for stacking R14 at the start of every subroutine, so that you cannot mistakenly nest deeper than you had intended. Of course there is a time overhead with the stacking operation. Stacking R14 takes 500 nanoseconds, while unstacking directly into R15 takes 800 nanoseconds. Avoiding this overhead on a frequently used piece of code can therefore pay dividends.

STACKING MULTIPLE REGISTERS

In the examples given so far, we have only used the stack to save the contents of R14, but any or all of the ARM's registers may be saved in this way. The only thing to remember is that you must unstack the same number of registers that you stack, otherwise all manner of problems will arise. By stacking multiple registers you can of course ensure that a given subroutine preserves the state of registers for the main program. Having fifteen user registers at your disposal when programming the ARM gives considerable flexibility, but you can still easily run out of registers. However, by using the stack you can temporarily free registers R0 to R12 for the duration of any subroutine. As an example, the subroutine below preserves registers R0, R6, R7, R8 and R14:

```
.main program
STMFD R13!,{R14}
BL subroutine
.
.
LDMFD R13!,{PC}
.
.
.subroutine
STMFD R13!,{R0,R6-R8,R14}
;subroutine code
LDMFD R13!,{R0,R6-R8,PC}
```

The important thing to note is that the registers stacked are the same as those unstacked, except for R14 which is replaced by PC to effect a return to Basic.

Subroutines and register stacking are thus easily accomplished if you follow the few simple examples given in the article. Next month we will look at shift and rotate operations, all performed without time overhead thanks to the magic of the ARM's Barrel Shifter. RU

U-CONNECT FROM MAGENTA RESEARCH

Ian Burley, News and Features Editor of Micronet, takes a long hard look at U-Connect, which was the first commercial comms package to be released for the Archimedes.

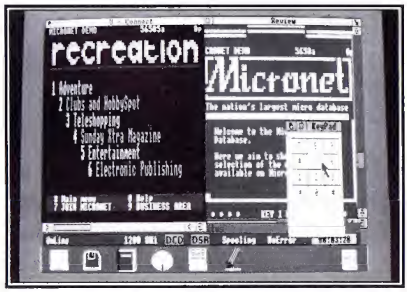
Magenta Research has one claim to fame with their U-Connect comms package - it was the very first commercial package to be released for the Archimedes. Magenta Research's main line of business is developing communications solutions for mini and mainframe systems, often running Unix. Thus Magenta had a ready-made library of core routines written in C which could be ported to Acorn ANSI C on the Archimedes. The bulk of Magenta's work on U-Connect has been in grafting their routines to Arthur 1.20 and the WIMP desktop.

In May, version 1.02 as tested here was released, and though this is the official current release, Magenta tell me that they are collating existing users' comments in order to release a further improved U-Connect in a couple of month's time. To be quite frank, an improved version is an absolute necessity in my view if Magenta want to compete successfully in the Archimedes communications market.

U-Connect uses the standard Archimedes WIMP desktop and some vibrant colours have been chosen to give the screen layout a bit of life. It could be argued that the choice of colours, i.e. lots of reds, and purples combined with brighter whites, yellows and cyans, are a bit hard on the eye. A petty point some might think, but it has been proven through behavioural research that bright colours hasten fatigue in front of VDU screens. The icon designs aren't very indicative of their function either, and on the whole look rather inexpertly put together.

While presentation might not win U-Connect any prizes, functionality is not at all bad. Specifications are pretty good with scrolling text teletype, VT52, VT100, and Prestel/Viewdata emulations on offer. ASCII, X-Modem, Kermit, and even Y-Modem file transfers are already available, with more planned. CET telesoftware in viewdata mode can also be downloaded. Kermit operation currently doesn't support multiple file transfers, and its implementation appears sluggish in response to commands.

Scrolling text emulations are offered in a full-width window with black text on a white background. In VT100 mode the host can invoke a 132 column display via an Escape sequence, though this is shown via a left-right scroll window instead of using one of the Archimedes' built in 132 column screen modes. A non-functioning 132 column icon option remains as a relic of an abandoned attempt at an alternative user-selectable 132 column display.



A small but handy feature missing from U-Connect is the ability to segregate incoming text from that being entered at the terminal. Comms packages which offer this windowed text mode make online multi-user game playing much easier as text you are typing won't get obliterated by text being simultaneously received.

File transfers are neatly handled by dedicated dialogue/status boxes where the filename of a file to be transferred can be entered and its progress monitored in bytes, blocks and time spent. Worth mentioning is the fact that files are not automatically saved to disc after a download. I personally believe that the priming of the filing system should be performed before the download starts rather than after, so that there is no delay in automatically saving the file to disc after the transfer is completed.

The viewdata emulation is best described as a compromise. Magenta have opted for a 40 column black background window, i.e. only half the width of the screen, and only the viewdata mosaic graphic character set has been re-defined. Separated graphics are not supported either, although double height is. Flashing colours are crudely implemented, and simple tests like Prestel's 'shaking hand' mailbox frame are poorly displayed.

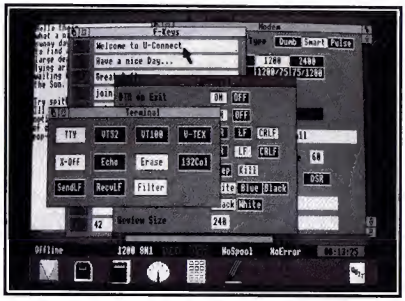
Editors for both the viewdata and text modes are mentioned in the on-screen help display, but not implemented. Therefore U-Connect is not capable of editing and compiling off-line mailboxes. Even on-line, you are faced with the problem of the viewdata emulation failing to keep up as the cursor traverses dynamically. To make many colour or graphic alterations in the preparation of a mailbox on-line requires much re-displaying of the frame.

A small numeric pad window is provided for keying the star (*) and hash (#) keys, and page numbers, but there is no facility for direct keying of routes in a frame with the mouse pointer.

U-Connect's modem support is fairly comprehensive, with drivers supplied for dumb modems, smart modems including Hayes types, and software pulse dialling modems such as the original Demon modem, or the related Magic modem, etc.

Smart modem support offers the usual options for modem initialisation, dial-up strings, and modem connect/disconnect status. Modem driver files are reconciled with the built-in telephone directory for automatic dial-up and log-on as required. No phone unit cost display is offered, though an on-line status window and clock can be clicked up. Audible beeps are emitted by U-Connect when a carrier is either detected or lost - quite a nice touch, and it can be switched off if you don't like it.

U-Connect can be booted up either by using Shift-Break in the usual fashion, or by using the WIMP desktop, but the user will have to modify the supplied !BOOT file to install the essential RS423 driver bug-fix module. Magenta's excuse for not making the installation of the fix automatic is that it isn't possible if the machine is in Basic - a lame excuse if ever I heard one. However, care should be taken with the !BOOT setup as Magenta have opted for ADFS macro commands and a proper disc name. I accidentally deleted the disc name from within one macro and virtually disabled the disc!



That about wraps up U-Connect. If it were the only terminal package around for the Archimedes I might grudgingly accept it. However, as it is, the package doesn't stand up to comparison with the likes of Hugo Fiennes' ArcTerm 6.01, or the standard-setting BEEBUG Hearsay package. The only really kind thing I can say about U-Connect is that it is relatively simple to use and I didn't have to scurry off to the 52 page manual all the time.

Magenta's Brian Smith has assured me that the next release of U-Connect, which will be available free to all existing registered users, will have major improvements including a completely new viewdata section. I await this with interest, but in the meantime the sad fact is that at almost £60, U-Connect is the most expensive Archimedes comms package and far from the best.

RU

Product	U-Connect Terminal Emulator
Supplier	Magenta Research 5th Floor, Amp House, Dingwall Road, Croydon CR0 9XA. Tel. 01-680 2585
Cost	£59.95 inc. VAT and p&p

Acorn's Kermit

Ian Burley reports on Kermit, the file transfer protocol, and in particular on Acorn's implementation for the Archimedes.

Kermit is the name of a file transfer protocol which is used widely for transferring files between different machines. It has its origins at the University of Columbia, USA. Why name a file transfer protocol after a silly green frog puppet? The Kermit User Guide (Sixth Edition, by Columbia University's Frand da Cruz) published here by Acorn fails to explain this, but does appear to cover virtually everything else you might want to know about this slightly mysterious file transfer utility. This is just as well, because for your £56 you are really paying solely for the manual - the actual Kermit software (supplied on disc by Acorn) is available free from its nominated distributors, and in the UK you can download Acorn versions of Kermit for various machines, including the BBC B and Archimedes, direct from the University of Lancaster.

Kermit has been around since 1981 and was developed at Columbia University for file transfers between their DEC System 20 mainframe and various CP/M micros around the campus. XMODEM and similar error checking protocols have been around much longer, but full implementations of Kermit don't simply provide an error checking protocol, but a complete easy-to-use environment for sending and receiving files between two computers.

Acorn's version of Kermit for the Archimedes was ported over from the old Acorn Cambridge Workstation (CWA), a National Semiconductor 32016 machine running Acorn's own in-house PANOS operating system. Very little effort has been made to tailor the Kermit C source code from the ACW to the Archimedes. In other words, the program runs very simply, totally avoiding the desktop, and Kermit commands must be typed into a simple prompt. A VT52 compatible terminal emulation is provided for communicating with the selected host machine or server, and Archimedes Kermit also has the basic tools to operate as an automatic server itself. In other words, another micro could attach or log on to your Archimedes and control the Archimedes' Kermit remotely. This is fine if you are connected up locally via a null modem cable, but Archimedes Kermit offers no built-in auto-answer modem support whatsoever.

These basic features mean that Archimedes Kermit is a very traditional implementation of Kermit, and one which Columbia University would probably readily recognise and be at home with. However, the plain fact is that the Archimedes' conversion is to say the least rudimentary (you can't even click on the Kermit program from the desktop to get going), and the fact that many embellishments like modem support are missing means that this offering is not really superior to the built-in Kermits of proprietary Archimedes comms software such as ArcTerm and BEEBUG's Hearsay (see RISC User Issue 9).

Hardened Kermit enthusiasts will no doubt find Acorn Kermit fascinating to get into, and certainly there is a lot to say about Kermit in general for which there isn't adequate room here, but the conclusion is that Acorn's package can't be recommended. Kermit is an excellent vehicle for file transfers (this article was edited on a BBC Master and transferred to an Apple Macintosh using Kermit). But if you really want to get hold of it, download it for free from Lancaster University - and I'm only repeating the advice of an official Acorn spokesman!

Product	Kermit
Supplier	Acorn Computer Ltd Fulbourn Road, Cherry Hinton, Cambridge CB1 4JN. Tel. (0223) 214411
Price	£56.35 inc. VAT.

RU

FAX Pack

FAX PACK FOR THE ARCHIMEDES

The Computer Concepts FAX PACK adds the power of the Archimedes computer to a FAX machine. It includes the following features:

A FAX modem podule that simply plugs into any Archimedes computer, and then into any standard telephone socket. It is possible to connect a conventional phone in parallel.

Software which allows character based output from a variety of word processors to be converted to a FAX file format, suitable for sending to any group 3 FAX machine. This can also convert SPRITES and other graphic files to a FAX format file. So it is possible to send screen dumps or text files, not only to other machines, but also to any group 3 FAX machines.

Auto-dial allows other computers or FAX machines to be dialled by the computer. Either a phone number or a name can be specified. Names are looked up in a phone number directory file to obtain the correct number.

FAX images may be viewed on screen, within a window, where they may be scaled, scrolled, cropped etc., and then saved again as a FAX file. Facilities are included for printing out FAX files on a variety of dot matrix printers, including the most common 9 and 24 pin versions.

The popularity of FAX machines has increased to such an extent during the last twelve months that they now outnumber TELEX machines. In today's successful business, a FAX machine is as vital a piece of office equipment as a telephone.

Since FAX modems work at 9600 bits-per-second, this allows direct file transfer at least 4 times as fast as the fastest conventional modems. Indeed any file can be automatically sent, **or obtained** from a remote machine.

The software automatically detects the type and group of machine dialled. It can also detect a suitably FAX equipped computer. If it other end, similar to this computer, then it can avoid the FAX protocols and transmit the files, error checked, directly to the remote machine.

Auto dial and auto-answer hardware which sends or receives any group 3 FAX, and saves on the current filing system as a suitable FAX file.

Most facilities are available from simple logical star commands, such as *DIAL Fred and *SendFAX file office.

FAX PACK INCLUDES:

The FAX podule for plugging into any Archimedes computer, (fitted with a back-plane) and software as described above.

There is also the option of a 200 dpi, A4 width, hand held document scanner, which plugs into the FAX podule. The scanner unit allows any printed document to be digitised and saved in a suitable FAX file format. In other words, this system provides an economic alternative to stand alone FAX machines with the added advantage of the Archimedes computer and associated software.

Price: £499 plus VAT or £799 plus VAT with the scanner.

Please phone for availability

(Access & Barclaycard accepted)

▲ Fax Pack Document Scanner

B.T. APPROVED

Computer Concepts Ltd

Gaddesden Place Hemel Hempstead Herts HP2 6EX Telephone 0442 63933

(Access & Barclaycard accepted)

Postbag

We welcome your letters for publication on our Postbag page whether they contain comments on the magazine and the Archimedes, technical queries or information for other readers.

BASIC BUGS

While converting a Basic program from the Master 128 to an Archimedes, I came across an interesting bug in Basic V. The problem occurred in a line similar to:

```
    10TIME=0:Q%=INKEY(300)-64:IF TIME>=
300 PRINT "TIME" ELSE IFQ%<-1 THEN 10
```

If this line is executed, and the Escape key is pressed within three seconds, the whole computer hangs up. This can be cured by splitting the line into two, and also, the problem does not occur if the '-64' is removed. This bug is specific to Basic V, and does not occur on the Master 128.

K. Gardner

Thank you to Mr Gardner for pointing this out. Obviously, this bug is very obscure, and unlikely to cause any major problems. However, it is always useful to be aware of such problems.

FASTER SPHERES

Having entered the listing from the Archimedes Visuals in Issue 8 to draw mode 15 spheres, I was rather shocked by how slow the program ran. I decided to try and improve the speed, and to my surprise this proved easier than I expected. Firstly, the procedure consists of two nested loops, and the variables P2% and D2 are both calculated inside the second loop, when they can in fact be removed outside that loop. Line 280 of the original ensures that both D1 and D2 are positive, when in fact they are both squared in the next line, making this unnecessary.

Once these changes have been made, the calculation is simplified to the point where the temporary variables P1% and P2% can be removed. Further, both D1 and D2 can be made integers, and the squaring of D2 can be moved outside the inner loop. Finally, the use of DIV in line 240 can be removed, as the integer arithmetic will perform the rounding. This results in the new procedure:

```
DEF PROCsphere(col%,rad%,L1%,L2%,pix%)
FOR Y%=rad% TO -rad% STEP -4
  A%=SQR(rad%*rad%-Y%*Y%)
  D2%=DEG ASN(Y%/rad%)-L2%
  D2%=D2%*D2%
  FOR X%=-A% TO A% STEP pix%
    D1%=DEG ASN(X%/rad%)-L1%
    C%=7.99-SQR(D1%*D1%+D2%)/14-RND(1)
    IF C%<0 C%=0
    GCOL 0,col%+(C% AND 4)*5.25 TINT
      (C% AND 3)*64
    PLOT 69,X%,Y%
  NEXT
NEXT
ENDPROC
```

Richard Davies

The reason why the published routine was so slow is that it draws the sphere pixel by pixel, calculating the shading required for each. In this context the 20% speed increase provided by Richard's amendments is most welcome. Our article on faster Basic in this issue gives more detailed guidance on improving the speed of programs.

EXEC ERRORS

I have been trying to enter EXEC files by using the *BUILD command, and then setting the file type to &FFE, as suggested in Hints and Tips of Issue 1. However, I seem to run into problems when trying to insert the '|' or '<' characters into a file. Can you help?

P. Gadd

Mr Gadd's problem is one that confuses many users - both experienced and novice. In order to allow control codes to be entered into a command line, the operating system reserves '|' as a special character. If '|' is followed by a letter, then it will have the same effect as pressing that letter with the control key. For example, |M in a command line will simulate the Return key being pressed. Similarly, '<' is also treated as a special character. It is used to enter operating system variables, or ASCII values, into a command line.

*The problem is, that when you try to enter these two characters into an EXEC file, the *BUILD command picks them up first and interprets them with their special meaning. The cure is to add an extra '|' before both '|' and '<'. In other words, '|' becomes '||', and '<' is '|<'. If the EXEC file is listed with *TYPE or *LIST, the extra '|'s will be shown, but when it is executed it will be interpreted correctly.*

RU

David Spencer rounds up the latest hints and tips for the Archimedes.

LOCATING THE MOUSE

Lee Calcraft

If you are using the mouse to select options in any program, you should find the following procedure very handy. Its purpose is to determine whether or not the pointer is located within any given rectangle on the screen. It is called with six parameters, as follows: the x and y co-ordinates of the pointer, the x and y co-ordinates of the bottom left-hand corner of the target rectangle, and its width and height respectively. It returns the value TRUE if the pointer is within the rectangle, and FALSE if it is not.

```
DEFFNisitthere(mx,my,destx,desty,
   destw,desth)=mx>destx AND
   mx<destx+destw AND my>desty
   AND my<desty+desth
```

BOOTING FROM THE DESKTOP

Geoffrey Waits

Quite a few pieces of commercial software which are started with Shift-Break will fail to boot at all when the Desktop is selected as the default language. This is because these programs try to *EXEC the !BOOT file, and the Desktop totally ignores the characters read from the file. One simply way around this, provided you have fast fingers, is to press Shift-Break normally, and as soon as the Desktop's pointer appears on the screen move it to the bottom right hand corner. Then, as soon as the icon bar is drawn, click on the 'Exit' icon, and the !BOOT file will then be executed. It must be stressed that these operations must be performed very quickly for this method to work.

The official way to allow a disc to auto-boot from within the Desktop is to make the !BOOT file an executable program. This can either be a Basic program, or some form of machine code routine, depending on the filetype. For Basic, SAVE "!BOOT" will set the correct filetype. The auto-boot option should then be set to RUN using the command *OPT 4,2.

MOVING FILES

David Pilling

One way of moving a file from one directory to another is to use the *COPY command, and include the 'D' option to delete the source file after it has been copied. This, however, can be quite slow, because the COPY command actually moves each byte of the file. A much better solution is to use *RENAME, which can be used to change the entire pathname of a file, rather than just its filename. For example, to move a file called DATA from the root directory to the sub-directory FINANCE, you could use:

```
*RENAME $.DATA $.FINANCE.DATA
```

This is much faster, because only the directory entries are changed; the actual file contents are untouched. This also means that the new file is in the same place on the disc, which can be useful in some circumstances. The only problem of this method is that you cannot use *RENAME to move a file between different discs, or even different filing systems, while *COPY will cope with both of these.

DEBUGGER BREAKPOINTS

Dr P. Borcherds

The Debugger module in Arthur provides a useful breakpoint facility. However, it can be tedious to discover the address at which a breakpoint should be set, and then to set it. The method given here simplifies the process. The first step is to insert the line:

```
DIM break(16):brk%=0
```

at the start of the source code program, and the function definition:

```
DEF FNbreak brk%+=1
[OPT pass:.break(brk%) BNV 0:]=""
```

at the end of the program. The variable 'pass' should be replaced by the name you have used for the pass counter. Obviously, you must ensure that the new variable names do not conflict with any already used in the source code.

To define a breakpoint, simply insert the line:

```
EQUS FNbreak
```

in the source code wherever you want to set a breakpoint. Up to 16 breakpoints may be set. The source code is then assembled in the usual way.

Finally, before the assembled machine code is called, the actual breakpoints must be set up. This is best handled by defining a function key to perform the job, thus:

```
*KEY 1 *BREAKCLR|MYFOR I%=1TO16:
   IFbreak(I%) OSCLI"BREAKSET"
   +STR$~break(I%): NEXT ELSE NEXT|M
   *BREAKLIST|M
```

If function key 1 is then pressed immediately after assembling the program, all the breakpoints will be set up and then listed.

RISC User Magazine Disc

October 1988

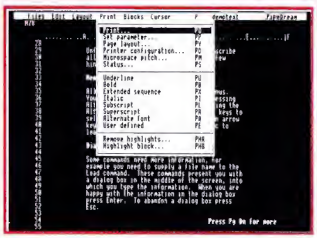
RISC USER DISC MENU Make accessing your discs easy with this enhanced version of the RISC User Disc Menu. The disc contains the complete source code.

DYNAMIC BOXING AND SPRITE GRABBER A useful procedure that allows you to select an area of the screen by using the mouse to drag a box around. The accompanying program lets you pick up part of a picture and use it as a paint brush.

ARCHIMEDES ANIMATION
Displaying a rotating molecule.The concept of using dual screens for smoother movement is also introduced.

ARCHIMEDES VISUALS
Two more graphics programs: an implementation of John Conway's Life and a mode 12 beach ball.

POINTER DEFINER
Design your own pointers. The disc includes some samples.



REAL TIME IMAGE SPINNER
A program that can distort and rotate an image.

RISC USER TOOLBOX
An extended version of last month's disc sector editor to allow specified files to be edited.

MOUSE CONTROLLED CURSOR
This clever module allows the mouse to control the cursor, and simulate key presses.

BONUS ITEMS

PIPEDREAM

From Colton software, producers of the Pipedream integrated word processor, comes this sample version of the package. Most of the features except for Save and Print are included, as is full documentation.

ALERION

A demonstration version of Alerion, the new 'shoot-em-up' game from Dabs press.

RISC User magazine discs are available to order, or by subscription. Full details of prices etc. are given on the back cover of each issue of RISC User.

HINTS & TIPS

WAITING FOR THE MOUSE

Lee Calcraft

In programs which make use of the mouse buttons, you need to take great care to flush the mouse buffer before taking any new reading . But you cannot simply use *FX21,9 to flush the buffer, because of the speed of response of the mouse. The way around the problem is to use a waiting loop to halt your program until the mouse really is clear. The following routine performs the task. Its calling parameter is the button number for which the routine is to wait. This will normally be zero (meaning "wait until no button is pressed"). But it could equally well be any number from 0 to 7. Additionally, if the procedure is called with -1 (or TRUE) as the parameter, the routine will wait for any non-zero condition.

```
DEFPROCmousewait(n)
REPEAT:MOUSE x%,y%,z%
UNTIL z%=n OR (n=-1 AND z%>0)
ENDPROC
```

The procedure does not return parameters specifically, since this adds complexity. But once it has been called, the result of reading the mouse can be found in x%,y% and z% (z% being the button number of the last reading). If your program required a user input from the mouse on any button, you might use the following sequence:

```
PROCmousewait(0)
PROCmousewait(-1)
```

The first call makes sure that the mouse is clear before checking for buttons. The second call waits until any button has been pressed. In effect this pair of calls gives the mouse equivalent of the keyboard's:

```
*FX15
z%=GET
```

RISC USER magazine

MEMBERSHIP

RISC User is available only on subscription at the rates shown below. Full subscribers to RISC User may also take out a reduced rate subscription to BEEBUG (the magazine for the BBC micro and Master series).

All subscriptions, including overseas, should be in pounds sterling. We will also accept payment by Connect, Access and Visa, and official UK orders are welcome.

RISC USER SUBSCRIPTION RATES		RISC USER & BEEBUG
£14.50	1 year (10 issues) UK, BFPO, Ch.I	£23.00
£20.00	Rest of Europe & Eire	£33.00
£25.00	Middle East	£40.00
£27.00	Americas & Africa	£44.00
£29.00	Elsewhere	£48.00

BACK ISSUES

We intend to maintain stocks of back issues New subscribers can therefore obtain earlier copies to provide a complete set from Vol.1 Issue 1. Back issues cost £1.20 each. You should also include postage as shown:

Destination	UK, BFPO, Ch.Is	Europe plus Eire	Elsewhere
First Issue	40p	75p	£2
Each subsequent Issue	20p	45p	85p

MAGAZINE DISC

The programs from each issue of RISC User are available on a monthly 3.5" disc. This will be available to order, or you may take out a subscription to ensure that the disc arrives at the same time as the magazine. The first issue (with six programs and animated graphics demo) is at the special low price of £3.75. The disc for each issue contains all the programs from the magazine, together with a number of additional items by way of demonstration, all at the standard rate of £4.75.

MAGAZINE DISC PRICES	UK	Overseas
Single Issue discs	£ 4.75	£ 4.75
Six months subscription	£25.50	£30.00
Twelve months subscription	£50.00	£56.00

Disc subscriptions include postage, but you should add 50p per disc for individual orders.

All orders, subscriptions and other correspondence should be addressed to:

RISC User, Dolphin Place, Holywell Hill, St Albans, Herts AL1 1EX.
Telephone: St Albans (0727) 40303
(24hrs answerphone service for payment by Connect, Access or Visa card)